

# NAVAL POSTGRADUATE SCHOOL

## Monterey, California



## THESIS

**FORMAL SPECIFICATION AND  
ANALYSIS OF A WIRELESS  
MEDIA ACCESS PROTOCOL**

by

Martin Scott Almquist

September 1995

Thesis Advisor:

Gilbert Lundy

Approved for public release; distribution is unlimited.

19960221 033

DTIC QUALITY INSPECTED 1

<b>REPORT DOCUMENTATION PAGE</b>			Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time reviewing instructions, searching existing data sources gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.				
1. AGENCY USE ONLY (Leave Blank)		2. REPORT DATE September 1995		3. REPORT TYPE AND DATES COVERED Master's Thesis
4. TITLE AND SUBTITLE Formal Specification and Analysis of a Wireless Media Access Protocol			5. FUNDING NUMBERS	
6. AUTHOR(S) Almquist, Martin Scott				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING/ MONITORING AGENCY NAME(S) AND ADDRESS(ES)			10. SPONSORING/ MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the United States Government.				
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.			12b. DISTRIBUTION CODE	
13. ABSTRACT (Maximum 200 words) The problem addressed by this research is to formally specify and analyze a proposed wireless network media access protocol. The protocol, named MACAW for Multiple Access Collision Avoidance Wireless, was described in ACM SIGCOMM Proceedings 94 Vol. 24 #4. The approach taken was to use the formal model Systems of Communicating Machines to develop a formal specification of the protocol. An initial specification was derived directly from the original proposal in order to reveal any unresolved problems. The formal specification was then refined to produce a more precise and unambiguous specification. The refined specification was used to analyze the protocol using system state analysis for properties such as liveness and deadlock. Liveness is the property of positive progression while deadlock is an undesirable property where a state is reached that cannot be left. The results are a specification of MACAW as originally proposed and a refined specification which provides an unambiguous understanding of the protocol. The analysis determined that the protocol is free of deadlock. Also presented are three new transitions between MACAW states, which were suggested by the analysis.				
14. SUBJECT TERMS Formal Specification, Wireless Communications, Media Access			15. NUMBER OF PAGES 96	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL	



Approved for public release; distribution is unlimited

**FORMAL SPECIFICATION AND ANALYSIS  
OF A WIRELESS MEDIA ACCESS PROTOCOL**

Martin Scott Almquist  
Major, United States Marine Corps  
B.S., University of Arizona, 1981

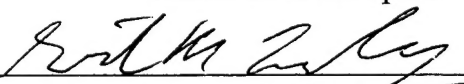
Submitted in partial fulfillment of the  
requirements for the degree of

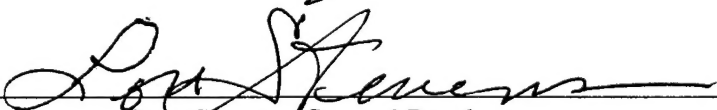
**MASTER OF SCIENCE IN COMPUTER SCIENCE**

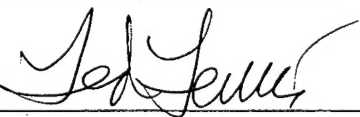
from the

**NAVAL POSTGRADUATE SCHOOL  
September 1995**

Author:   
Martin Scott Almquist

Approved by:   
Gilbert Lundy, Thesis Advisor

  
Lou Stevens, Second Reader

  
Ted Lewis, Chairman,  
Department of Computer Science





## ABSTRACT

The problem addressed by this research is to formally specify and analyze a proposed wireless network media access protocol. The protocol, named MACAW for Multiple Access Collision Avoidance Wireless, was described in ACM SIGCOMM Proceedings 94 Vol. 24 #4.

The approach taken was to use the formal model Systems of Communicating Machines to develop a formal specification of the protocol. An initial specification was derived directly from the original proposal in order to reveal any unresolved problems. The formal specification was then refined to produce a more precise and unambiguous specification. The refined specification was used to analyze the protocol using system state analysis for properties such as liveness and deadlock. Liveness is the property of positive progression while deadlock is an undesirable property where a state is reached that cannot be left.

The results are a specification of MACAW as originally proposed and a refined specification which provides an unambiguous understanding of the protocol. The analysis determined that the protocol is free of deadlock. Also presented are three new transitions between MACAW states, which were suggested by the analysis.



# TABLE OF CONTENTS

I.	INTRODUCTION .....	1
A.	WIRELESS NETWORKING .....	2
B.	FORMAL SPECIFICATION .....	3
C.	ANALYSIS .....	3
D.	ORGANIZATION OF THIS THESIS .....	4
II.	WIRELESS NETWORKS .....	5
A.	GENERAL .....	5
B.	ISSUES IN WIRELESS NETWORKING .....	5
1.	Wireless versus Wired Networks .....	6
2.	Multiaccess Networks .....	6
3.	Multiple Access Protocols .....	7
C.	MACA .....	10
1.	Hidden and Exposed Nodes .....	10
2.	MACA .....	11
III.	MACAW .....	15
A.	GENERAL .....	15
B.	MACAW FEATURES .....	15
1.	MACAW Description .....	15
2.	Message Exchange Procedure .....	16
3.	Backoff Algorithm .....	19
4.	Protocol Message Contents .....	19
C.	PHYSICAL LAYOUT OF MACAW IMPLEMENTATION .....	19
1.	Inner-Office LAN .....	19
2.	Physical Layer .....	20
IV.	FORMAL SPECIFICATION .....	23
A.	FORMAL DESCRIPTION .....	23
B.	SYSTEMS OF COMMUNICATING MACHINES .....	23
C.	SPECIFICATION OF MACAW .....	24
1.	Local and Shared Variables .....	25
2.	Predicate action table and State diagram .....	27
3.	Ambiguities encountered in specifying MACAW .....	28
D.	REVISED MACAW SPECIFICATION .....	30
1.	Refinements in MACAW_2 .....	33
V.	ANALYSIS .....	35
A.	GENERAL .....	35
B.	ANALYSIS PROGRAM .....	35
1.	User Written Program Units .....	36
2.	Input File .....	37
C.	ENCODING MACAW FOR ANALYSIS .....	38
1.	Number of Machines and Variables .....	38
2.	Timer Settings and Backoff .....	39

3.	Transition Naming .....	40
4.	QUIET State .....	40
D.	RESULTS OF ANALYSIS .....	40
1.	2 Machines .....	40
2.	3 Machines .....	41
3.	4 Machines .....	42
E.	IMPROVEMENTS SUGGESTED BY ANALYSIS .....	42
1.	Receive DATA from WFDS State .....	43
2.	Loop Transition in WFDS State .....	44
3.	Loop Transition in WFCTS State .....	45
VI.	CONCLUSIONS .....	47
A.	FORMAL SPECIFICATION .....	48
B.	ANALYSIS .....	48
C.	IMPROVEMENTS TO MACAW .....	49
D.	SUGGESTIONS FOR FURTHER WORK .....	49
	LIST OF REFERENCES .....	51
	APPENDIX .....	53
	INITIAL DISTRIBUTION LIST .....	87

## I. INTRODUCTION

Wireless networking has experienced rapid growth in recent years as more and more applications are found that can benefit from wireless networks. Applications from inner-office LANs to WANs exist as well as private, commercial and military uses. One concern shared by all wireless networks is media access control (MAC). MAC protocols form a layer of any network model and are particularly important for wireless networking since the media, be it radio, light, or sound must be shared amongst the devices in the network. A good MAC protocol will resolve media contention both in the interests of fairness and network throughput.

This thesis presents a formal specification and analysis of a wireless Medium Access Control (MAC) protocol. The formal specification was developed using the Systems of Communicating Machines model. The specification and analysis led to several suggestions for improvement in the protocol. The wireless MAC protocol was developed at Xerox's Palo Alto Research Center (PARC) by a team of designers: Alan Demers, Scott Shenker, Lixia Zhang and Vaduvur Bhagavan [1]. The protocol, termed MACAW, for Multiple Access Collision Avoidance Wireless, was developed from the Multiple Access Collision Avoidance (MACA) protocol proposed by Karn for use in packet radio networks [2]. MACA and MACAW both were designed the wireless network environment. MACA was proposed as a media access protocol for packet radio that would address the hidden and exposed node problems common in wireless networks. It would not use the Carrier Sense Multiple Access approach to determine when the media was clear for transmissions but would explicitly query the intended recipient. The query would be in the form of a Request to Send (RTS) message addressed to the intended recipient. If the intended recipient was able to receive the traffic it would respond with a Clear to Send (CTS) message. After receiving the CTS message the original machine would then transmit its data. [3] and [4] argue for a RTS-CTS-DATA message exchange procedure for wireless applications. MACAW was developed specifically for use in a single channel wireless LAN

network structure being developed at Xerox Corporation's Palo Alto Research Center Computer Science Laboratory. Building on the MACA, MACAW uses a similar message exchange but adds three new messages. The new messages are DATA SEND (DS), REQUEST to send REQUEST TO SEND (RRTS), and ACKNOWLEDGEMENT (ACK). The full MACAW message exchange procedure is RTS-CTS-DS-DATA-ACK. The key to understanding the functioning of MACAW lies in understanding this message exchange procedure.

## **A. WIRELESS NETWORKING**

Recent years have seen a proliferation in wireless networking and mobile computing. Wireless digital communication is seen as a key enabling technology for mobile computing since it offers an escape from the tyranny of cables and wires. However, wireless communication is not a panacea to all communication needs, it has restrictions in contrast to conventional hard wired networks while offering several advantages over traditional "wired" communications. Advantages of wireless networks include: lessened requirements for supporting infrastructure, quicker network installation, and a more flexible network architecture. Among the restrictions or limitations of wireless networks are: problems associated with transmission range, local congestion, and interference that comes with sharing the same medium amongst many machines.

Wireless and wired networks will play complimentary roles: wireless networks will extend connectivity to mobile and remote users as well as users in dynamic networks wherein users enter and leave the network on an random basis. Wired networks will continue to move the bulk of data and provide "backbone" connectivity between groups of wireless users and the rest of the network. In this scenario wireless LANs will play a major role in giving groups of users mobility within a "local area" e.g. an office, warehouse, or some other relatively limited area.

Military applications of wireless networking abound. Wireless LANs are envisioned to provide ground forces with tactical data communications for command posts and small

unit communications [6]. Wireless Wide Area Networks (WANs) also find applications ranging from satellite communications to tactical communications uses among ground units.

## **B. FORMAL SPECIFICATION**

MACAW is formally specified and analyzed using a model called Systems of Communicating Machines [6]. This model has been used to analyze several communications protocols e.g. FDDI, CSMA/CD, etc. The Systems of Communicating Machines (SCM) model is used to assist in the describing and analyzing communications protocols. Each machine in an SCM model is defined as a finite state machine with variables. Communication between machines is done through the use of global or shared variables. Local variables are used to maintain the state information for each machine. In general, the goal of a formal specification is to analyze the protocol for 'liveness' properties which is defined as the ability to make positive progress. An additional benefit is gained from the formal specification in terms of identifying any ambiguities in the protocol.

## **C. ANALYSIS**

Once the formal specification has been developed the protocol can be analyzed. Global state and system state reachability analysis are methods used to analyze communications protocols. Global reachability is commonly used in conjunction with the Communicating Finite State Machine (CFSM) model. Global reachability suffers from a combinatorial explosion of states as the number of machines grows. System reachability analysis is normally used with the SCM model. It does not suffer the combinatorial explosion of numbers of states that the global analysis does and improves on some other aspects of the global reachability analysis method. Even with system state reachability analysis the number of possible system states can grow unwieldy for manual analysis. Consequently, it is useful to have an automated capability to conduct either a system or global state reachability analysis. The analysis in Chapter V is based on a computer program that does the analysis.



#### **D. ORGANIZATION OF THIS THESIS**

This thesis is organized in the following manner: Chapter I is an introduction to the thesis and briefly outlines the topic and procedure. Chapter II discusses wireless networks in general and the particular aspects of media access that make wireless networks fundamentally different from wired networks in this respect. It provides a motivation for the importance of MAC protocols and their formal specification. MACA, the conceptual ancestor of MACAW is presented along with a discussion of the hidden and exposed node problems that led to the formulation of MACA and in turn MACAW. Chapter III describes the MACAW protocol and the implementation details of the PARC LAN. Chapter IV contains the formal specification of MACAW. Two specifications are presented, the first is an initial specification based on the material in [1], and the second is a refined specification resolving the ambiguities discovered in developing the initial specification. Chapter V contains the analysis of MACAW based on the formal specification. The results of the analysis of the protocol and how the formal specification was translated into analysis code are presented. Several different configurations were analyzed starting with a simple two machine transmitter and receiver and ending with 4 machines modeling the exposed and hidden node performance of the protocol. Also presented in Chapter V are several improvements to the protocol that were recognized during the analysis.

## **II. WIRELESS NETWORKS**

### **A. GENERAL**

This chapter introduces some of the relevant issues in wireless networking. The need for multiple access to the media in a wireless network is explained as are some general features of wireless versus 'wired' communications. Several multiple access protocols are briefly discussed along with some of their advantages and disadvantages. Finally, MACA, the ancestor of MACAW is outlined. In the context of MACA the wireless problem of hidden and exposed nodes is presented.

Superficially wireless networking is much the same as regular networks. Nodes in the network exchange information via links between nodes. Both kinds of networks can be modeled as graphs where each machine or station is represented as a node in the graph and its links with other nodes as edges between the respective nodes. Issues such as congestion, message queueing, access control, are common to both. In many cases wireless networks are extensions of regular networks thereby providing mobility to the user.

In general, wireless communications will have lower data transmission rates and will be more susceptible to interference than standard networks. The lower transmission rates stem from two factors: first the theoretical data rates for radio frequency vary with frequency but even in the Gigahertz radio frequency range maximum data rates only approach that of a typical Ethernet e.g. 10 Megabits per second (Mbps). The second reason for lower throughput in wireless networks is the higher noise and interference found in wireless media particularly radio media that lead to higher error rates and consequently more time spent retransmitting data.

### **B. ISSUES IN WIRELESS NETWORKING**

Unlike hard wired networks where link-to-link communication is relatively free from concerns such as transmit power, signal propagation, and link distance, wireless networking must pay special attention to these needs in order to communicate successfully.

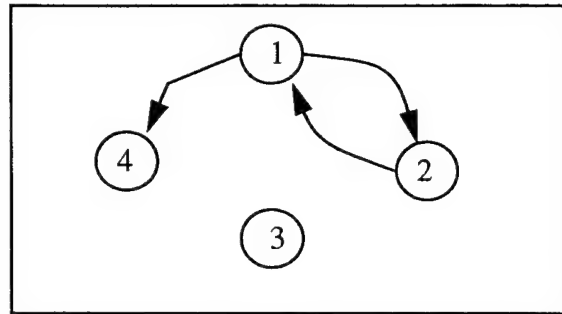
Wireless communications between various machines will require a mutually agreed upon sharing of the radio frequency spectrum. Various schemes have been proposed to share the medium between multiple machines. These include Carrier Sensing Multiple Access (CSMA), CSMA/ Collision Detection (CSMA/CD), Time Division Multiple Access (TDMA), Frequency Division Multiple Access (FDMA), and various Code Division Multiple Access (CDMA) techniques also known as Spread Spectrum. The goal of these methods is to allow multiple devices to access and use the media in some sort of disciplined way rather than every device transmitting whenever it has some data to transmit. Transmitting without regard to other traffic inevitably leads to collision, delays and congestion.

### **1. Wireless versus Wired Networks**

In contrast to hardwired networks where the physical link between two nodes can be constructed in such a way so as to allow only those two nodes and no others access to the link, wireless links share no such useful characteristics. In their media access characteristics, wireless networks of all types are similar to LANs in that they share the same media and hence must use the media with a minimum of interference with other nodes if useful work is to be done. Furthermore, factors that do not impact the conventional network such as terrain, locations, user mobility all have an effect on the wireless network. As result of these factors a wide variety of network configurations are possible and these configurations will change randomly as network stations move, background noise changes, etc. Figure 1 depicts a network where one station (node 1) can hear node 2's transmissions and node 2 can hear node 1. Node 4 can hear node 1 but node 1 cannot hear transmissions from 4. Node 3 can neither hear nor be heard by any of the other nodes. Contrast this with a typical hardwired Ethernet where every network station can hear every other station.

### **2. Multiaccess Networks**

A communications network can be thought of as a collection of nodes with arcs between the nodes representing the communications link in a manner similar to Figure 1.



**Figure 1: Wireless Configurations**

This link might physically consist of wire, co-axial cable or fiber optic cable. Each link in this network carries a signal that is a combination of the transmitted signal and whatever ambient noise exists on the link. In networks where multiple devices share the same media (typically satellite communications or local area networks e.g. ethernet, token ring or token bus) the received signal at any station is the sum of all the transmitted signals and the ambient noise level. Various methods, known as protocols, have been suggested to resolve the problems of sharing the same media between different stations. Because LANs by their very nature share the media, early wireless networks used media access schemes similar to those employed by LANs. The next section briefly describes some of the more common multiple access protocols beginning with probably the most common: CSMA.

### **3. Multiple Access Protocols**

#### ***a. CSMA and CSMA/CD***

A station using Carrier Sense Multiple Access (CSMA) first listens to the media to determine if another transmission is in progress. If a transmission is in progress the station waits. If no transmission is in progress, the station is free to transmit its own messages. The Collision Detection part of CSMA/CD works as follows: upon transmitting a message each station monitors the frequency to detect any other signals. By comparing the difference between what it knows the message to be with what is sensed on the frequency (medium) it is able to detect collisions. A problem that arises with CSMA and CSMA/CD is that the

machine only detects interference at its site. It has no way of knowing whether the message at the intended target site is going to be interfered with. In order to detect problems at the receiver the sending machine waits some predefined period of time to receive an acknowledgment from the target machine. If an acknowledgment is not received then the machine retransmits the message. This sequence is the same whether the target machine is off the air, busy receiving a message from a third machine, or transmitting a message of its own. It should also be noted that CSMA and CSMA/CD require specialized hardware to detect the presence of a carrier. This is one feature MACAW does not require e.g. specialized carrier sense hardware.

#### ***b. TDMA***

Time Division Multiple Access varies from CSMA in that each station is assigned a period of time, known as a "a slot" in which to transmit any information it has. The slot is generally assigned to an individual station by a master or base station which is responsible for distributing timing information to network stations. Slotted Aloha used this type of approach to increase bandwidth usage over the original Aloha protocol. Since timing information must be distributed, the protocol is essentially a centralized protocol depending on the central station for the timing information. Without a centralized source of timing information each station would drift off its true slot (due to small differences in computer clock time) and eventually begin interfering with other stations. Normally this central source is a master station in the network that uses its time as the global time for the entire network. Outside sources of time information are available such as the Global Positioning System but these sources introduce another system to the network and remain centralized in nature. A possible inefficiency exists in the case where a station has nothing to transmit during its slot but no way of giving up its slot to another station. Reservation schemes have been proposed e.g. the DQDB protocol that attempt to over come this by reserving slots with the master station. Any protocol depending on centralized information is vulnerable to single point failure and moreover almost precludes master station mobility

since the master station usually is positioned to give the maximum possible communications span. However, some exceptions to master station mobility do exist, one is the U.S. Navy's Naval Tactical Data System (NTDS). It uses a master station that queries other stations for traffic.

*c. FDMA*

Frequency Division Multiple Access allows multiple stations to be on the air at the same time by dividing the available frequency spectrum between stations. In this it is somewhat similar to frequency multiplexing wherein one station transmits several different communications threads (voice conversations, data, or video) by dividing the threads among the frequency bandwidth it has available. In terms of mobility and adaptability FDMA uses a prearranged division of frequencies and is not designed for dynamic adaptation to new network topologies. A version of FDMA that lends more mobility to the application is one wherein a master station dynamically assigns frequencies to other stations within its transmission range (known as a cell). This approach provides mobility to the user stations but is still vulnerable to single point failure at the master station. This particular type of FDMA is in fact the scheme applied in cellular telephone networks.

*d. CDMA (Spread Spectrum)*

Code Division Multiple Access, commonly known as Spread spectrum, "spreads" the signal orthogonally throughout the usable spectrum between users so that they may use the same spectrum simultaneously without interference. Spread Spectrum communications have received much attention of late in wireless communications circles since it has many characteristics valuable for communicating in the type of electronic environment inherent to mobile communications. It is relatively resistant to jamming or interference, and allows frequency reuse by simply assigning a different spreading code. However, it also must either preassign codes to using stations or a master station must assign the spreading codes to stations as they enter the network. In the case of preassigned codes the network is no longer able to accept new stations or in the case of a master station assigning codes the

network becomes vulnerable to single point failure at the master station. Another factor of note is that stations with different spreading codes cannot communicate directly with each other. Either they must somehow exchange spreading codes and use a single code to communicate or communicate through a third station.

### C. MACA

Karn first proposed MACA for use in packet radio networks [2]. MACA uses a RTS-CTS-DATA packet exchange and binary exponential backoff for media access resolution after a collision has occurred. Table 1 depicts the message types used in MACA. Karn

Table 1:

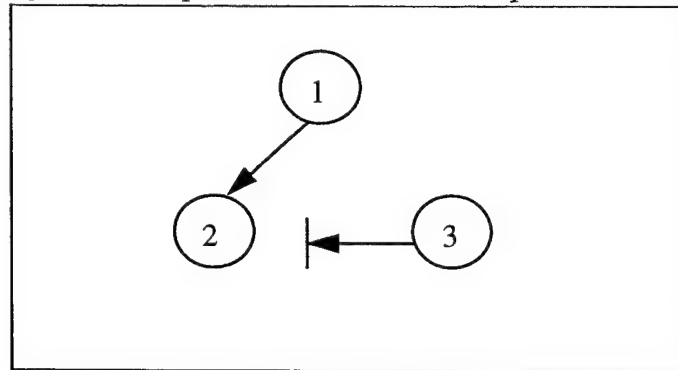
Name	Meaning
RTS	Request to Send
CTS	Clear to Send
DATA	Data message

observed that the CSMA approach in packet radio networks suffered from two problems: hidden nodes and exposed nodes.

#### 1. Hidden and Exposed Nodes

According to Karn [2] an inherent problem with wireless networks is what is known as hidden and exposed nodes. A hidden node is a situation where a node (3) is 'hidden' from a node (1) because node 1 cannot hear node 3's transmissions. An exposed node is a related situation in which a node (2) is exposed to node 1's transmissions (node 1's transmissions destructively interfere with node 2's reception) when node 2 is attempting to receive a transmission from node 3. Figure 2 depicts both situations. Node 1 is shown transmitting and consequently interfering with node 2's reception of 3's transmission to 2. Note that since node 3 is hidden from node 1, Carrier Sensing will not avail node 1 of any useful information when it begins to transmit. This is because Carrier Sense only provides

information about the media at the node doing the sensing. It does not provide any information about the media at other nodes. Furthermore using a Carrier Sense approach does not give node 2 any ability to inform node 1 that it is receiving a transmission from 3. It was from reasoning about this problem that Karn developed the motivation for MACA.

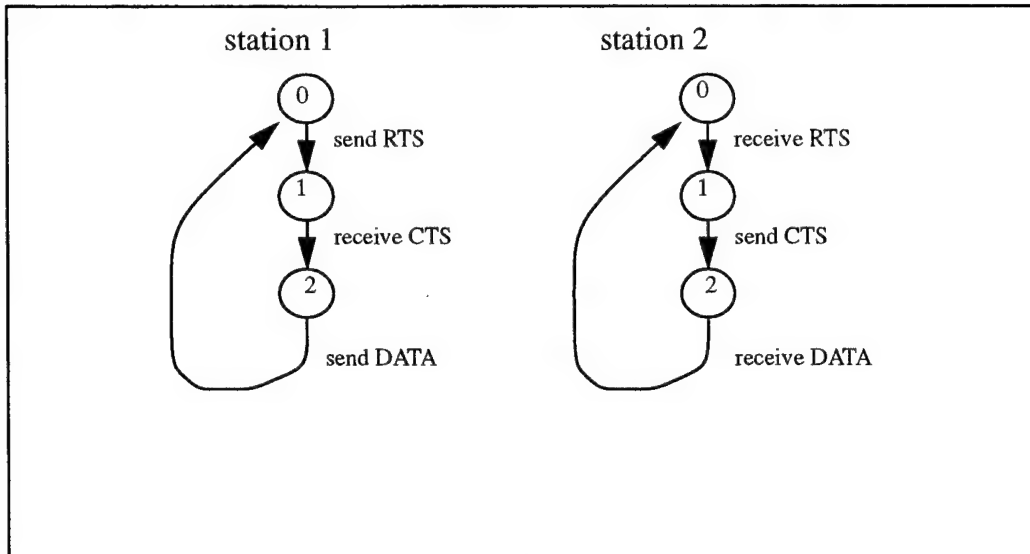


**Figure 2: Hidden and Exposed Nodes**

## **2. MACA**

From the hidden and exposed node problem Karn reasoned that if a node could inform other stations within its transmission range when it was either about to transmit or receive that much of the problem stemming from hidden and exposed nodes would be alleviated. His protocol, MACA, uses a RTS-CTS-DATA message exchange procedure. MACA uses a backoff algorithm known as Binary Exponential Backoff. MACA works in the following way. A station (1) wishing to transmit a message to another station (2) first transmits a RTS message to 2. The RTS message informs 2 that 1 has a DATA message for 2. Upon receiving the RTS message station 2 immediately sends a CTS message back to 1. The CTS message informs 1 that it is all right to send the DATA message. Any other station overhearing either the RTS or CTS message will defer its transmissions for an amount of time sufficient for the DATA message to be passed from 1 to 2. The deferral time is determined from a field in the RTS and CTS message that contain the length of the DATA message. Figure 3 depicts a state diagram for two stations using MACA. It represents a one way data transfer from station 1 to 2.





**Figure 3: MACA State Diagram**

By transmitting the RTS message, station 1 informs all nodes within reception range that it is about to transmit data to 2. Thus all the nodes receiving this defer and the exposed node problem is resolved. The CTS message similarly resolves the hidden node problem by informing all nodes within reception range of 2 that it is about to receive a message from 1. Other nodes hearing the CTS message defer until the DATA message is received. Therefore 2 is not subject to destructive interference from another node's transmission while receiving node 1's DATA message.

Before sending an RTS message a station will wait for a random amount of time. This time is determined by randomly choosing a number between 1 and the backoff counter value. This number is then multiplied by the slot length where slot length is the length of time it takes to transmit a RTS or CTS message (both are the same length). Normally, the only collisions in MACA will be collisions between RTS messages. Since an RTS message will be a fraction of the size of a DATA message, less time is spent in collisions than if the protocol simply transmitted the DATA message without the RTS or CTS messages. Collisions with CTS messages will not normally take place since the CTS message is sent immediately upon receipt of a RTS message. To collide with a CTS message a third station

would have to finish its wait period (the period before transmitting a RTS message) at the same time that the CTS message was sent. This also implies that the third station did not receive the RTS message. In the unlikely event that a collision does occur the subsequent DATA message would also be lost and the sending station would eventually start the process over again.

Note that it is possible for collisions to happen with other messages if either the RTS or CTS messages are not received by all the stations that can interfere with the transmission of the DATA message. This might take place for instance if intermittent noise is present that would prevent reception of the RTS or CTS by a node other than 1 or 2. Usually, assuming symmetry of reception and transmission would preclude this circumstance. However, given the nature of wireless communications, assuming symmetry between transmitter and receiver only applies in noise-free situations. Noise-free environments will not normally be found in the type of operating environment envisioned for mobile computing. As a result it is important that media access protocols work in noisy environments as well as the simpler situation of noise-free.



### **III. MACAW**

#### **A. GENERAL**

This chapter provides a brief discussion of the significant aspects of MACAW taken from [1]. MACAW was developed by a team at Xerox Corporations's Palo Alto Research Center (PARC). The team's work is based on a single channel radio LAN under development at Xerox PARC. Their analysis of media access protocols led them to propose a new protocol termed MACAW. MACAW improves on the media access methods of its ancestor, MACA, and uses a significantly different backoff algorithm for collision resolution. The improvements hope to achieve greater efficiency and better use of location dependent congestion knowledge.

#### **B. MACAW FEATURES**

##### **1. MACAW Description**

MACAW as presented in [1] is composed of eight states and six different messages. The eight states are IDLE, CONTEND, Wait for Clear to Send (WFCTS), Wait for Acknowledgment (WFAck), Wait for Data Send (WFDS), Wait for Data (WFData), QUIET, and Wait for Contend (WFCONTEND). The messages are: RTS, CTS, DS, DATA, ACK, and RRTS. All messages but the DATA message are 30 bytes long. The DATA message is 512 bytes long. The transmission time of the 30 byte message packet is the "slot" time used for setting the random timer through which media access is scheduled. The advantage gained in using so many control messages is the relative size of the data message and control messages e.g. 512 bytes versus 30 bytes. Also while not completely clear in [1] it seems that a machine may transmit multiple DATA messages for every RTS-CTS exchange. Otherwise the utility of including a DATA size parameter in the RTS, CTS

and DS messages seems pointless. Table 2 presents the MACAW message types for ease of reference.

Message	Meaning	length
RTS	Request to Send	30 bytes
CTS	Clear to Send	30 bytes
DS	Data Send	30 bytes
DATA	Data message	512 bytes
ACK	Acknowledgement	30 bytes
RRTS	Request to send an RTS	30 bytes

Table 2: MACAW Message Types

## 2. Message Exchange Procedure

MACAW uses a RTS-CTS-DS-DATA-ACK message exchange procedure. An RRTS message is provided to re-initiate communication when machine 1 receives a RTS message while in the QUIET state from machine 2. In order to describe the message exchange procedure the following is an example procedure involving four machines. Machine 1 has data to send to machine 2. Machine 3 and machine 4 are not directly involved in the procedure but are included to demonstrate how the transitions to the QUIET and WFCONTEND states take place and the use of the RRTS message. Figure 5 in Chapter IV graphically depicts the state diagram used in the following example.

### a. Machine 1

When machine 1 is in the IDLE state and has a message for machine 2 it transitions to the CONTENTEND state and sets a random timer an integer number of slot times between 1 and BO where BO is the value of the Backoff Counter. When the timer expires machine 1 transmits a RTS message and set a timer sufficient for machine 2 to respond with a CTS. It then transitions to the WFCTS state.

When machine 1 receives the CTS from machine 2 it transmits back-to-back a DS and DATA message. It sets a timer sufficient to receive an ACK from machine 2. It then transitions into the WPACK state. If instead machine 1 receives an ACK from machine 2, it will return to the IDLE state and clear message from its out buffer. Receiving an ACK message in the WFCTS state will happen if machine 1 had previously transmitted the DATA message to machine 2 but had not received the ACK message even though machine 2 had successfully received the DATA message.

Upon receiving the ACK message from machine 2 while in the WPACK state, machine 1 clears its output buffer of the related data message and transitions to the IDLE state.

If machine 1 does not receive the appropriate message from machine 2 in either the WFCTS or WPACK state its timer will eventually expire. When the timer expires in either state machine 1 returns to the IDLE state and begins the procedure anew by setting the random timer and transitions to the CONTEND state.

If machine 1 receives a RTS while it is in the CONTEND state it transmits a CTS message to the originator of the RTS and transitions to the WFDS state. At the same time it sets a timer sufficient to allow receipt of the DS message.

***b. Machine 2***

When machine 2 receives the RTS from machine 1 it transmits a CTS message, sets a timer sufficient for the DS message from machine 1 to be received and transitions into the WFDS state. If machine 2 has received machine 1's message in an earlier exchange it transmits an ACK.

From the WFDS state if machine 2 receives a DS message from machine 1 it sets a timer sufficient to receive the DATA message and transitions to the WFDATA state.

When machine 2 receives the DATA message it transmits an ACK message and transitions to the IDLE state.

If machine 2 does not receive the appropriate message from machine 1 while it (machine 2) is in the WFDS or WFDATA its timer expires and transitions machine 2 back to the IDLE state where it will wait for machine 1 to restart the communication.

***c. Machine 3***

When machine 3 receives a message not addressed to itself it transitions to the QUIET state. It can do this from any of the other 7 states. When it transitions to the Quiet state it sets a timer that determines how long machine 3 remains in the QUIET state. The value of the timer varies depending on which type of message machine 3 received. If machine 3 received the RTS from machine 1 to machine 2 its timer is set to allow for the CTS message to be received by machine 1. If machine 3 received the CTS from machine 2 it sets its timer to allow the DS and DATA message to be received by machine 2. Finally, if it received the DS message from machine 1 it sets the timer to allow the DATA and ACK messages to be exchanged. When the timer expires in the QUIET state machine 3 transitions back to the IDLE state.

If machine 3 receives an RTS while it is in the QUIET state it transitions to the WFCONTEND state. When the timer set in the transition to the QUIET state expires in the WFCONTEND state machine 3 sets a random timer between 1 and BO slot times long and transitions to the CONTEND state. When the random timer expires in the contend state machine 3 will transmit a RRTS message to machine 4 and transition to the IDLE state.

***d. Machine 4***

If machine 4 transmits a RTS message to machine 3 while machine 3 is in the QUIET state, machine 4 will timeout from the WFCTS state and return to the IDLE state and then reenter the CONTEND state as it still has data for machine 3. If while machine 4 is in the IDLE state it receives machine 3's RRTS message it transmits a RTS message to machine 3, sets a timer sufficient to receive machine 3's CTS and transitions to the WFCTS state.

### **3. Backoff Algorithm**

MACAW uses a different backoff algorithm than the Binary Exponential Backoff of MACA as it was felt by MACAW's developers that BEB oscillated too rapidly. Instead the developers determined to increase the value of the backoff counter by a multiplicative factor of 1.5 for every collision and decrease it by 1 for every success. The term coined for this backoff algorithm was Multiplicative Increase and Linear Decrease (MILD).

### **4. Protocol Message Contents**

Message types RTS, CTS, DS, ACK and RRTS are all 30 bytes long. Pertinent data carried in these messages are Destination Address (DA), Source Address (SA), message type (RTS, CTS, etc.), Exchange Sequence Number (ESN), length of DATA, local backoff, and remote backoff. The ESN is used by machines to keep track of messages it receives from other machines. Two ESNs are maintained per other machine, one for messages going to the other machine and one for messages coming from the machine. When a new transmission from machine 1 to machine 2 is initiated machine 1 increments the value of the outgoing ESN. When machine 2 receives the RTS if the ESN has not been incremented it transmits an ACK message otherwise it transmits a CTS. The local backoff and remote backoff fields are included to distribute deferral information. Each machine maintains a local and remote backoff value for each machine it is communicating with. The remote backoff is the backoff value of the remote station (machine 2) as estimated by machine 1. The local backoff is the value of the backoff of maintaining station (machine 1) as estimated by the remote station (machine 2).

## **C. PHYSICAL LAYOUT OF MACAW IMPLEMENTATION**

### **1. Inner-Office LAN**

MACAW is implemented on a single channel radio LAN at Xerox PARC. The goal of implementing MACAW was to develop a media access protocol for use in PARC's wireless infrastructure and explore basic performance and design issues regarding wireless

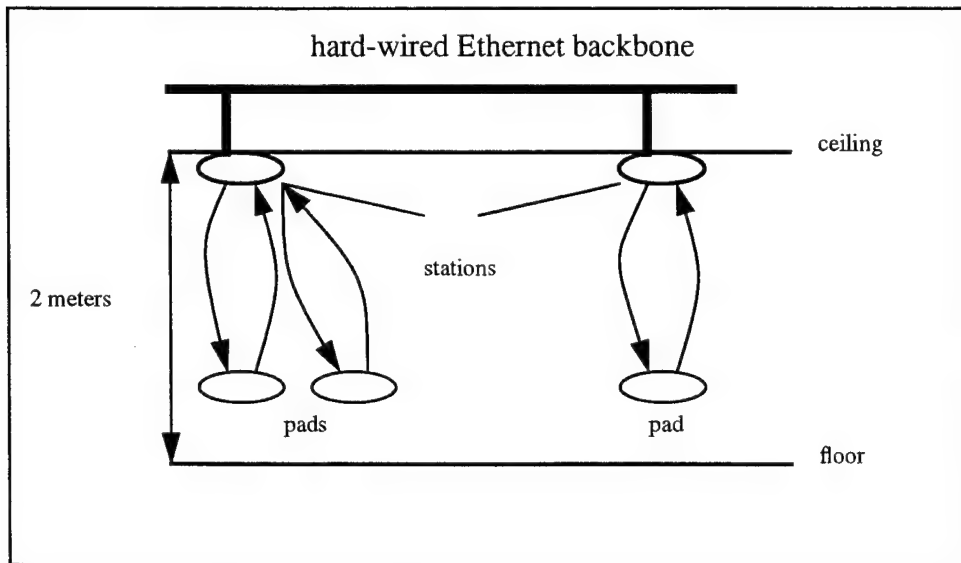


media access protocols. The MACAW LAN provides connectivity in an inter-office setting. Communication takes place between 'base stations' and 'pads'. The base stations are installed in the office ceiling and are connected to each other via a conventional Ethernet. Pads are custom built portable computing devices that connect to base stations via radio communications. One base station may have several pads with which it communicates while a pad will only communicate with a single base station. Unlike a fully functional LAN, there is no pad to pad communication in the PARC implementation. Pads may move from 'cell' to 'cell', where a 'cell' is the communication range of a given base station.

## **2. Physical Layer**

The Xerox PARC wireless LAN is built on a physical layer that functions in the 3 to 4 megahertz frequency range and utilizes PARC's 'near field' radio technology. The low frequency eliminates multi-path effects frequently found in a cluttered indoor environment. There is a single 256 kilobyte per second channel that all devices share and all devices transmit at the same power. An apparent advantage of using 'near field' technology is that the signal strength decays very rapidly and limits the transmission range to about 3 meters. This combination produces a cell with very sharply defined boundaries and about 6 meters in diameter. By strategically placing the base stations, inter-cell interference is minimized. Figure 4 shows a schematic diagram of a MACAW LAN.

An assumption made by the developers regarding the radio technology is that all communication is symmetric, that is that if machine 1 can hear machine 2 then machine 2 can hear machine 1. Most of their testing and simulation was done in a noise-free environment. In Chapter V this assumption is ignored in developing the analysis of MACAW.



**Figure 4: Schematic of a MACAW LAN**



## **IV. FORMAL SPECIFICATION**

### **A. FORMAL DESCRIPTION**

Formal description techniques are used to develop formal specifications of communications protocols. Formal description results in a precise, unambiguous description of the protocol in the form of a specification. Several different models have been proposed for use in formal description of protocols, among them Communicating Finite State Machines and Petri Nets. In this thesis the formal model used is known as Systems of Communicating Machines (SCM) [6]. The Systems of Communicating Machines model was designed to formally describe and define network protocols. As a result of its formality, the SCM model also contributes to a precise, unambiguous understanding of the protocols behavior and lends itself to automated analysis of the protocol. The next section briefly describes the paradigm of Systems of Communicating Machines. Section C develops an initial specification of MACAW. This initial specification was developed to provide a general understanding of the protocol as described in [1]. The rigor of a formal specification revealed various ambiguities and contradictions. No attempt was made to correct these problems in the initial specification as it was desired to point out the problems revealed using the formal model rather than attempting to textually describe problems with the original work. Section D develops a more precise and comprehensive specification that resolves the ambiguities and contradictions revealed in developing the specification in Section C.

### **B. SYSTEMS OF COMMUNICATING MACHINES**

The Systems of Communicating Machines model is briefly described in this section. More details can be found in [6].

A system of communicating machines is represented as an ordered pair  $C = (M, V)$ , where  $M = \{m_1, m_2, \dots, m_n\}$  is a finite set of machines and  $V = \{v_1, v_2, \dots, v_n\}$  is a finite set of

shared variables with two subsets  $R_i$  and  $W_i$  for each machine. Subset  $R_i$  is the set of read access variables for machine  $m_i$ . Subset  $W_i$  is the set of write access variables for  $m_i$ .

Each machine is defined by a tuple  $m_i (S_i, s_0, L_i, N_i, t_i)$ , where  $S_i$  is a finite set of states,  $s_0$  is a state in  $S_i$  designated the initial state of  $m_i$ .  $L_i$  is a finite set of local variables.  $N_i$  is a finite set of names, where each name is associated with a unique pair  $(p, a)$  and  $p$  is a predicate on the variables of  $L_i$  and  $R_i$  and  $a$  is an action on the variables  $L_i, R_i, W_i$ .  $t_i$  is a transition function which maps from the states and names of  $m_i$  to the states of  $m_i$ . system state tuple is a tuple of all machine states. An equivalent system state is said to be one where each machine is in the same state as its corresponding machine in the other system state. A global state is the system state tuple together with all the variables local and shared.

### C. SPECIFICATION OF MACAW

An initial specification of MACAW is developed in this section using the SCM model. This specification was developed directly from [1] with a minimum of changes or clarifying assumptions. This was done to illustrate any problems with the protocol as presented in [1]. Doing this serves to separate the work of this thesis from the work presented in [1]. The pseudo-code in Appendix 2 of [1] proved useful in developing the specification of the protocol.

The complete formal specification of MACAW will consist of the local and shared variables, state machine and predicate action table. The formal specification depicted in Figure 5 and Table 2 was developed from the pseudo code rules in Appendix B of [1]. Each device is modeled as a machine which maintains the local variables: inbuffer, and outbuffer. Communication between machines is achieved through the use of the shared variables that model the information on the radio channel. Radio channel information corresponds to the message header fields.

In order to provide a specification with the widest utility some MACAW-specific attributes are not modeled. Example: the exchange sequence number (ESN) at each pad need only be a single integer for incoming messages and a single integer for outgoing messages

since the pad only communicates with the base station and there is consequently need for only one ESN. However, in the general case of a fully capable LAN, pad to pad communication would be warranted. To accomplish this would require each pad to maintain a table of ESNs, two for each machine (two would be required if traffic flowed in both directions). For this specification, each machine communicates with only one other machine. In a like manner, each machine was given only one DATA message to send so that in buffers and out buffers could be represented as a single variable rather than a composite data type i.e. an array or record. The backoff and deferral rules are included in the predicate action for completeness but they will be considered abstract functions and as such not analyzed.

### **1. Local and Shared Variables**

Each machine maintains a set of local variables that constitute part of the state of the machine at any given instance. The shared variables that provide the means of inter-machine communications correspond to the information contained in the header fields of the messages described in Chapter III, e.g. the RTS, CTS, DS, RRTS, and ACK messages. These shared variables will be annotated as Channel\_\*, where the wild card represents the particular variable type: message type, address, ESN, Backoff, etc.

Predicate rules for communications between machines can be accomplished by maintaining shared variables for message type and destination address. Shared variables constituting the other message header fields are not used to determine the value of the predicate. This is due to each machine communicating with only one other machine.

Each machine maintains various local variables in order to manage state information. The local variables of each machine are inbuffer, and outbuffer. T Channel\_\* can be used to model the physical propagation of radio signals by controlling which machines write and read to which Channel\_\* variables. Outbuffer is of the same type as msg but will only hold DATA messages.

Channel\_type inbuffer and outbuffer are all of the same type and represent the type of MACAW message being transmitted i.e RTS, CTS, DS, DATA, ACK, and RRTS. Transmission takes place by writing the message type to the shared variable channel. Channel\_DA is used to represent the destination address of the transmitted message. By reading and writing to different channel and channel\_DA variables it is possible to model the behavior of different configurations of transmission and reception. Each machine is thought of as being able to receive transmissions only from machines within a certain distance and transmit to machines (not necessarily the same) within a certain distance. To receive messages each machine reads only from one channel and channel\_DA variable. This is the same as thinking of a machine as being able to receive messages that reach its antenna a certain power level above the ambient noise level of the spectrum. In transmitting a machine may write to multiple channel\_type and channel\_DA variables. Each of these variables in turn is read by a single machine. Note that in trying to accurately model possible transmission and reception configurations of radio communications it is necessary to allow for the possibility that machine 1 may be able to hear machine 2 while machine 2 may not be able to hear machine 1 as well as the more commonly postulated and simple configuration of symmetry where both machines can hear the other machine's transmissions. Additionally, the ability to model exposed and hidden node conditions is accomplished by using multiple shared variables. In contrast, hard wired LANs would use a single channel that each machine wrote to for transmissions and read from for receptions.

## 2. Predicate action table and State diagram

This section depicts the state diagram and Predicate-Action Table (PAT) for the initial MACAW specification. Figure 5 depicts the state diagram for MACAW.

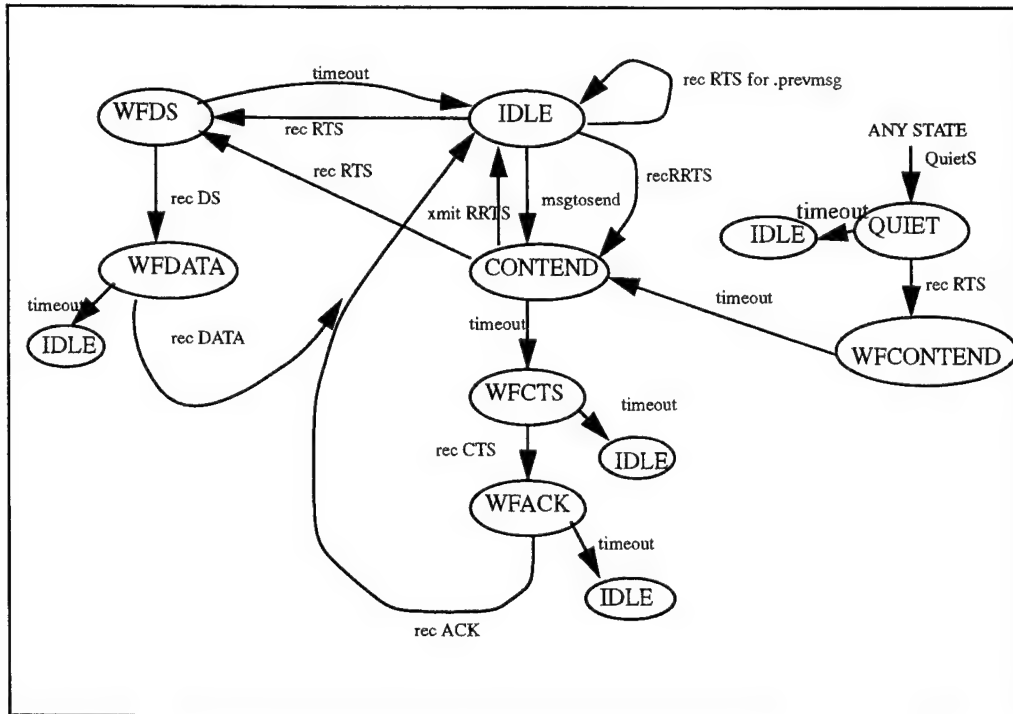


Figure 5: Initial MACAW State Diagram



Table 3 depicts the Predicate-Action Table for the initial MACAW specification.

<i>Transition</i>	<i>predicate</i>	<i>action</i>
1. msgtosend	outbuffer $\neq$ E	set timer, state:= CONTENTD
2. rec RTS	channel.DA = i	inbuffer:= channel, xmit CTS set timer
3. rec CTS	channel = i	inbuffer:= channel, clear timer xmit DS, xmit DATA, set timer
4. rec DS	channel = i	inbuffer:= channel, set timer
5. rec DATA	channel = i	inbuffer:= channel, clear timer, xmit ACK
6. rec ACK	channel = i	inbuffer:= channel clear timer, clear out buffer
7. rec RTS	rec RTS for prev msg	inbuffer:= channel, xmit ACK
8. rec RTS	channel.DA = i	xmit CTS, set timer,
9. rec RTS	channel.DA = i $\wedge$ state = QUIET	
10. rec RRTS	channel.DA = i $\wedge$ state = IDLE	xmit RTS, set timer, state:= WFCTS
11. timeout	state = WFCONTEND	set timer
12. timeout	state = CONTENTD	if enter(CONTEND) = IDLE xmit RTS, state:= WFDS else xmit RRTS, state:= IDLE
13. timeout	state $\neq$ CONTENTD $\parallel$ WFCONTEND	reset timer
14. Quiet	channel.DA $\neq$ i $\wedge$ channel = RTS	set timer = length (CTS + c)
15. Quiet	channel.DA $\neq$ i $\wedge$ channel = DS	set timer = length(DATA + ACK + c)
16. Quiet	channel.DA $\neq$ i $\wedge$ channel = CTS	set timer = length(DATA + c)? ACK
17. Quiet	channel.DA $\neq$ i $\wedge$ channel = RRTS	set timer = length(RTS + CTS + c)? ACK

TABLE 3: MACAW Predicate-Action Table for Machine i.

### 3. Ambiguities encountered in specifying MACAW

Attempting to formally specify MACAW revealed certain ambiguities and contradictions. From the pseudo-code in Appendix 2 of Ref [1] it was apparent that the

predicate column would require state information to determine which transition applied. The verbal description of the transitions and actions sometimes used the same name for different transitions or actions while using the current state as the means of determining which action or transition was to be executed.

*a. Transition Timeout*

The transition timeout is different depending on which state the machine is in when the timeout occurs. Additionally, the action upon timeout differs depending on which state and what is in the channel. For instance depending on which state the machine was in determined whether the transition **timeout** returned to the IDLE, CONTEND or WFCTS state.

*b. State Contend*

From state CONTEND Figure 5 has three different transitions. Two of these transitions depend not only on the timeout but also on what was the previous state. If contend was entered from IDLE then a RTS is transmitted but if CONTEND was entered from WFCONTEND then a RRTS is transmitted and the machine returns to the IDLE state. This would require another local variable to determine from which state CONTEND was entered.

*c. State Quiet*

The QUIET state can be entered from any of the other seven states. However, when entering the quiet state the timer is set to a different values depending on which type of message was in channel. Basically, the timer is set to allow the other two machines enough time to exchange messages. The time allowed for this varies depending again on what was in the channel. If a RTS was heard then the timer is set to allow a responding CTS to be sent. If a CTS was heard then the timer is set to allow the DATA message to get through. A DS message causes the timer to set for a DATA and ACK message. Lastly, a RRTS message causes the timer to set to set for the entire message RTS-CTS exchange. It

was noted that while hearing a DS message allowed enough time for the ACK to be received, a CTS only set the timer for the DATA message to be received, not the ACK message.

*d. Set Timer*

Only one timer is used in [1] and as such it is used for timing a variety of events that take different times. The action column in Table 1 reflects this fact by the notation  $\text{set timer} = \text{length} (\text{message type} + c)$ . The constant  $c$  is device and environment dependent factor for things such as propagation delay, radio key time, etc.

*e. Destination address field*

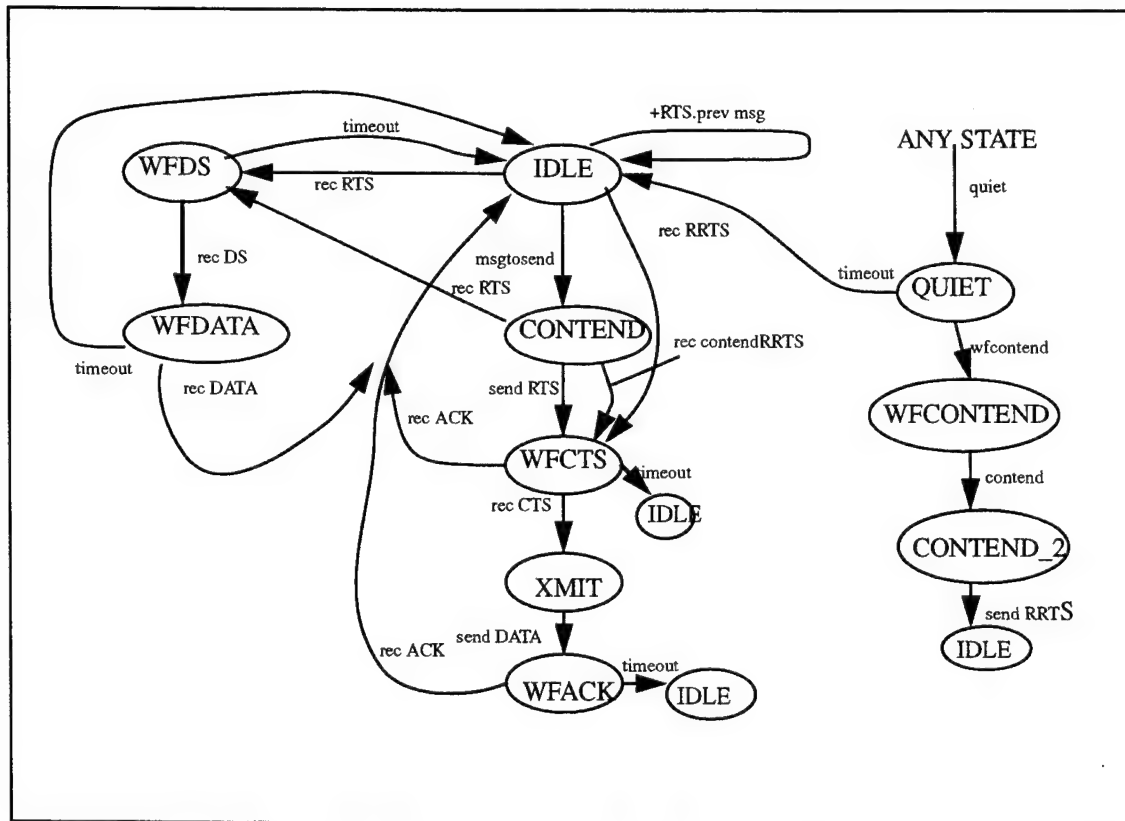
Explicitly mentioned in [1] was rule that whenever machine  $i$  received a message not addressed to itself then it set a timer and transitioned to the QUIET state. Implicit in this is whenever machine  $i$  can make one of the transitions 1 -11 then not only does machine  $i$  concern itself with the contents of the message type but must also check the destination address. This is not clear in [1] and consequently was translated verbatim into the specification of MACAW.

## **D. REVISED MACAW SPECIFICATION**

This section presents the MACAW specification as revised during the course of specification development. This revised specification (referred to henceforth as MACAW\_2) attempts to resolve the ambiguities uncovered in Section 3 and also present improvements to the protocol. The shared and local variables remain basically the same. Shared variables are `channel_type`, `channel_da`, `esn`, `channel_my_backoff`, `channel_local_backoff`, and `channel_retry` amount. Local variables are `inbuffer`, `outbuffer`, `backoff`, `my_backoff`, `local_backoff`. Figure 6 and Table 4 depict the formal specification for MACAW\_2.

The primary differences from the original specification in Section C are: two new states were added to the specification, several transitions were renamed for greater clarity

and timer values were specified or assumed. The transition to the quiet state from the WFCONTEND state was removed. This transition did not make sense in light of the fact that a machine would still have to wait in the CONTENTD state which would afford it the opportunity to move to the QUIET state.



**Figure 6: MACAW\_2 State Diagram**

The different timeout transitions were renamed to remove the ambiguity regarding which specific timeout was in question.

Table 4 depicts the Predicate-Action Table for MACAW\_2.

<i>Transition</i>	<i>predicate</i>	<i>action</i>
1. msgtosend	outbuffer $\neq$ E	set random timer(1, BO)
2. rec RTS	channel = RTS $\wedge$ channel.da = i	inbuffer:= channel, channel:= CTS, set timer (DS + c)
3. rec CTS	channel = CTS $\wedge$ channel.da = i	inbuffer:= channel, clear timer, channel:= DS $\wedge$ channel.da = j
4. send DATA	True	channel:= DATA, channel.da = j, set timer (ACK + c),
5. rec DS	channel = DS $\wedge$ channel.da = i	inbuffer:= channel, set timer (DATA + c)
6. rec DATA	channel = DATA $\wedge$ channel.da = i	inbuffer:= channel, clear timer, channel:= ACK, channel.da = j
7. rec ACK	channel = ACK $\wedge$ channel.da = i	inbuffer:= channel, clear timer, clear outbuffer
8. rec RTS.pre-vmsg	channel = RTS $\wedge$ inbuffer = DATA $\wedge$ channel.da = i	channel:= ACK, channel.da = j
9. wfcontend	channel = RTS $\wedge$ channel.da = i	
10. rec RRTS	channel = RRTS $\wedge$ channel.da = i	channel:= RTS, set timer (CTS + c)
11. send RRTS	timer expires	channel:= RRTS, channel.da = j
12. contend	timer expires	set random timer(1, BO)
13. timeout	timer expires	clear inbuffer,
14. send RTS	timer expires	channel:= RTS, set timer (CTS + c)
15. quiet	channel.DA $\neq$ i $\wedge$ channel = RTS	set timer = (CTS + c)
16. quiet	channel.DA $\neq$ i $\wedge$ channel = DS	set timer = (DATA + ACK + c)
17. quiet	channel.DA $\neq$ i $\wedge$ channel = CTS	set timer = (DS + DATA + ACK + c)
18. quiet	channel.DA $\neq$ i $\wedge$ channel = RRTS	set timer = (RTS + CTS + c)
19. rec contend-RRTS	channel.DA = i $\wedge$ channel = RRTS	clear random timer, set timer(RTS + c)

TABLE 4: MACAW\_2 Predicate Action for Machine i.

## **1. Refinements in MACAW\_2**

### ***a. XMIT State***

State XMIT was added to the specification so that the receiver side WFDS and WFDATA states would have corresponding states on the transmitter side. Since the timer in the WFDS state is set to allow a responding DS message to arrive and then another timer in the WFDATA state to allow the DATA message to arrive it would be logical to have corresponding states in the transmitter. Originally, when a CTS message was received by machine *i* it transmitted a DS and DATA message and went to the WFAK state. Adding XMIT allows the automated analysis to analyze the case where the DS is received but not the DATA message. If both messages are sent during the transition to the WFAK state the analysis will be unable to resolve the previous case.

### ***b. Contend\_2 State***

State CONTEND\_2 was added to remove the confusion resulting from the original specification calling for different transitions from CONTEND depending on what had been the previous state. As mentioned this would require another local variable to maintain the previous state knowledge. Instead another state was added.

### ***c. Timeout Transitions***

The three different transitions of MACAW were named timeout. These have been changed as follows: the timeout from the WFCONTEND state is now send RRTS, the timeout from the IDLE state is now called contend, and the timeout from the other states remains timeout as the same action happens when this transition is executed regardless of from which state it is executed.

### ***d. Transition rec contendRRTS***

This transition (number 19 in PAT) was added to the CONTEND state so that if a RRTS from machine *j* is received by machine *i* in the CONTEND state, machine *i* will send a RTS to machine *j*. This transition is important because it is unlikely that machine *i*

will ever be the QUIET state when a RRTS is received. While explicit timing information was not included in [1], it appears that machine i will timeout from the WFCTS, return to the IDLE state, and transition immediately to the CONTEND state while machine j is still in the WFCNTEND and CONTEND\_2 states. The absence of a transition from the CONTEND state upon receiving a RRTS appears to be a contradiction.

## **V. ANALYSIS**

### **A. GENERAL**

A major benefit of formal specification of protocols or computer programs is the potential for automated generation and analysis. Automated code generation was not part of this thesis, however, automated analysis of the protocol was. This automated analysis was conducted using the basic program developed in [7]. Bulent Bulbul, LTJG Turkish Navy, developed the analysis code as part of his thesis research at the Naval Postgraduate School. The next section outlines the basics of the analysis program as developed by Bulbul. Section C covers the actual code and the decisions made in representing MACAW specifics required to run MACAW with Bulbul's program. Section D contains the results of the analysis. The basic results are that the protocol is free of deadlock and, given two assumptions, the protocol will make positive progress: a property referred to as 'liveness'. The two assumptions that are key to liveness are 1) that a communication link does exist between the two machines in the absence of noise and other network traffic and 2) network traffic and noise will allow the two machines to communicate. Several improvements to MACAW were suggested as a result of the analysis. The improvements were in the form of additional transitions to allow more rapid media access after a timeout occurred and to enable the DATA message receipt without receipt of the DS message.

### **B. ANALYSIS PROGRAM**

Bulent Bulbul, LTJG Turkish Navy, developed an Ada program for the automated analysis of communications protocols. The program is able to conduct both System State Analysis and Global State Analysis based on the Systems of Communicating Machines Model. The SCM model is used to conduct the analysis. The analysis is done by constructing a directed graph of states and the transitions between states. The states are either System States or Global States depending on the type of analysis being conducted.



A System State is the tuple representing the individual machine states while a Global State is a tuple representing the individual machine states as well as the values of all local and shared variables.

## **1. User Written Program Units**

For each different protocol the user must encode the rules of the protocol in the form of predicates and actions. Additionally, definitions basic to the protocol are encoded as is the data and format for output. Each of these (predicates, actions, definitions, and output) take the form of a separate Ada program unit. The source code for these program units is located in the Appendix. Once compiled into executable code, the program reads a user generated input file to build the directed graph of states and transitions between states.

### ***a. Package Definitions***

Basic to the encoding of the communications protocol for Bubul's program was an Ada package named Definitions. This definitions package contains basic definitions for use by other program units and as such must be one of the first units compiled. In this package transition and machine types are defined as well as number of machines, buffer type, and global variables. Transitions are an Ada enumerated type and are distinct for each machine e.g. for machine 1 its transitions are msgtosend1, timeout1, send\_RTS1, etc. Table 4 shows the translation of transitions for machine 1 from the transitions of Table 3 to those used in the analysis program. Transitions for other machines are similar.

Buffer type is used to represent message types for each machine. For machine 1 DS1, RTS1, CTS1, DATA1, ACK1 and RRTS1 are the specific message types.

Machine types were represented by an Ada record type. The record for each machine was composed of two buffer type variables: out\_buff\* and in\_buff\* where the wildcard character represents the machine number.

Global variables were used to model the communications between machines. Rather than represent the entire MACAW header data it was necessary to only represent

the message type (a buffer type variable) and the destination address (an integer corresponding to the machine number).

***b. Analyze Predicates and Action Procedures***

The same file contained the separate Ada procedures for analyzing the predicates of each machine and the action procedure for all machines. Each Machines had it own separate procedure called *Analyse\_Predicates\_Machine\** where the wildcard character again represents the particular machine. The Action procedure contained the actions for each transition of the model. As a result it was necessary to define separate transitions for each machine so that for example receiving a RTS at machine1 would cause different actions than receiving a RTS at machine 2.

***c. Global Output Procedure***

The procedure *output\_Gtuple* is a user defined procedure that outputs the global state of the system. Since varying the number of machines, global variables, or local variables will change the global state this procedure must be modified when any of the previous items are changed. This procedure is used only when a global state analysis is conducted. The system state analysis used a predefined procedure to output the system state information.

**2. Input File**

Once compiled, the analysis program reads an input file. The four different input files used for the analysis of MACAW are contained in the Appendix. The input file can be modified to run different numbers of machines and transitions between states and different initial states as long as all the machines, states, or transitions are previously defined in the Definitions package and/or the *Analyse\_Predicates\_Machine\** procedure. States were numbered instead of named since the original program used only numbered states. Table 4

depicts the correlation between the states from Figure 6 and the numbered states of analysis program.

Table 5:

Figure 6 states	Analysis Program states
IDLE	0
CONTEND	1
WFCTS	2
XMIT	3
WFAK	4
WFDS	5
WFDATA	6
QUIET	7
WFCONTEND	8
CONTEND_2	9

### C. ENCODING MACAW FOR ANALYSIS

To encode the Predicate-Action table and state diagram for MACAW some basic network configuration decisions were made. These decisions were made so that the full range of states and transitions could be explored while keeping the overall number of states and transitions manageable.

#### 1. Number of Machines and Variables

Four machines were encoded with the complete MACAW\_2 specification. Machines 1 and 2 were constructed to have machine 1 send a DATA message to Machine 2 and once Machine 2 received that message it would send a DATA message to Machine 1. Machine 3 and machine 4 were constructed to explore the QUIET state transitions by using their transmissions to interfere with the transmissions between machines 1 and 2.

The shared variables modeling the radio media were represented by 'CHAN\*' variables. The wildcard character was used represent which machine read those particular variables. Specifically, CHAN \* and CHAN\*\_DA were the variables used to represent message type and destination address for machine \*. Each machine read its own set of CHAN\* variables e.g. machine 1 reads from CHAN1 and CHAN1\_DA. Each machine would write to one or more set of channel variables. This was done so that the shared media and hidden-exposed node behavior could be analyzed. Allowing a machine to read from only one set of media variables while writing to one or more variables models the characteristics of a radio medium wherein not all machines will necessarily be able to hear each other and any single machine will hear only those transmissions that reach its antenna a certain strength above the channel noise level.

Each machine maintained three variables: in\_buff\* for input buffer, out\_buff\* for output buffer, and state. DATA messages for transmission were stored in out\_buff\*. Incoming message types were stored in in\_buff\* and overwritten by succeeding messages. By storing the incoming DATA message in the input buffer (in\_buff\*) the machine could determine whether it had received this message previously by testing whether in\_buff = DATA when a RTS message type was received. This was done rather than maintain a table of exchange sequence numbers (ESN) to determine previous receipt of a DATA message.

## **2. Timer Settings and Backoff**

Timer settings were not explicitly modeled in conducting the analysis. This accomplished two things. First, it obtained analysis for the widest range of possible timer values by building the graph based on possible transitions rather than the most desirable. Desirable in this case being the transitions that would take place if all machines were had fairly similar values for timer settings. Second, it reduced the overall complexity of the analysis by reducing the number of shared and local variables that would be maintained. In this case, variables would not be required for backoff, local backoff, and remote backoff since time would not be explicitly modeled. If it was then these values

would all have to be maintained by each machine and also maintained as shared variables between machines. As mentioned in Chapter IV this would require a table of local and remote backoff variables for each machine that Machine *i* was communicating with.

### **3. Transition Naming**

As outlined Section B.1.a, transitions were individually named for each machine. This was a result of all the transitions being contained in the program's action procedure. In order to differentiate between receive a RTS at machine1 and a receive RTS at Machine 2 the transitions were named rcv\_RTS1 and rcv\_RTS2 respectively.

### **4. QUIET State**

Machines 3 and 4 were used to analyze the QUIET state and transitions to subsequent states (WFCONTEND, CONTEND\_2). By causing Machine 3 to write to Machine 1's CHAN\* variables machine 1 would transition into the QUIET state since the CHAN\_DA will be 3 vice 1. Once in the QUIET state, Machine 1 could then transition either back to the IDLE state or the CONTEND\_2 state via the WFCONTEND state.

## **D. RESULTS OF ANALYSIS**

Three basic configurations were analyzed with the automated analysis. All three used the same code as in the Appendix. The number of machines was varied by using three input files with two, three and four machines. The input files are contained in the Appendix for reference. Not all machines executed all the possible transitions of MACAW. This was unnecessary as long as at least one of the machines executed the relevant transitions that particular configuration. For example, in the two machine analysis, none of the quiet state transitions were executed for the simple reason that there was no other machine causing interference. In none of the cases analyzed did deadlock occur.

### **1. 2 Machines**

Machine 1 and machine 2 were run in the two machine analysis. Both began in the IDLE state and machine 1 had a message to send to machine 2. Once machine 2 received

the DATA message from machine 1 it sent a DATA message to machine1. (Machine 2's action upon receiving a DATA message included writing a DATA type message to its own output buffer: see Appendix). No deadlock occurred and the total number of system state generated was 69. The corresponding global state analysis generated 275 states. One artificial deadlock occurred in the global state analysis upon receipt by machine 2 of the ACK message from machine1. The deadlock happened as a result of clearing the output buffers when an ACK message was received. Essentially, both machines received their respective DATA messages and no more data remained to be transferred, thus causing a 'deadlock'.

## **2. 3 Machines**

Machine 3 was added to the configuration for two machines in order to analyze the quiet state behavior of machine 1 only. This was done by causing machine 3 to write to CHAN1 and CHAN1\_DA. When machine 1 received a CHAN1\_DA other than 1 it transitioned to the QUIET state. An extra action was added into the action for the quiet transition to reset the CHAN1 variable to E (Empty). This had to be done due to the way in which machines wrote to other CHAN\* variables but cleared only their own CHAN\* variables. Without the clearing action (resetting to E) the machine would continue to read traffic in its CHAN\* even though the transmitting machine had finished.

The analysis generated 676 system states without any deadlocks. The artificial deadlock that took place in the case of the two machine analysis did not occur since a third machine existed that still had transitions available when machine 1 and machine 2 had no more transitions left. Machine 1 executed transitions to the QUIET state from all other states and successfully returned to the IDLE state. Both machines 1 and 2 successfully passed their respective DATA messages to each other. Machine 3 only executed the msgtosend, send\_RTS and timeout transitions since there was not a machine responding to machine 3's transmission.

### **3. 4 Machines**

Machine 4 was added to the analysis in order that machine 3 would have a machine to exchange data with and thereby test the quiet state transitions for all the other message types besides the RTS type. (Only the RTS message was sent by machine 3 in the three machine analysis above since it would never receive a CTS to proceed further). Machine 4 also interfered with machine 2's reception as the 'send\_RTS4' action in the action procedure wrote to machine 2's CHAN2 and CHAN2\_DA variables as well as CHAN3 and CHAN3\_DA. This in turn caused machine 2 to transition into the quiet state and the subsequent states. This configuration analyzed the hidden and exposed node behavior of the protocol.

The analysis generated 2003 states of which 1580 were unique states. The analysis program hit a preset check point at the 2003rd state and was exited at that point rather than continue with what would have been repetitive analysis. No deadlocks were encountered and the DATA messages were successfully exchanged between machine 1 and machine 2. An additional note on the configuration, machine 4 did not return a DATA message to machine 3. Neither machine 3 nor 4 executed any quiet state transitions as there were no machines writing to their CHAN1 and CHAN1\_DA VARIABLES besides themselves.

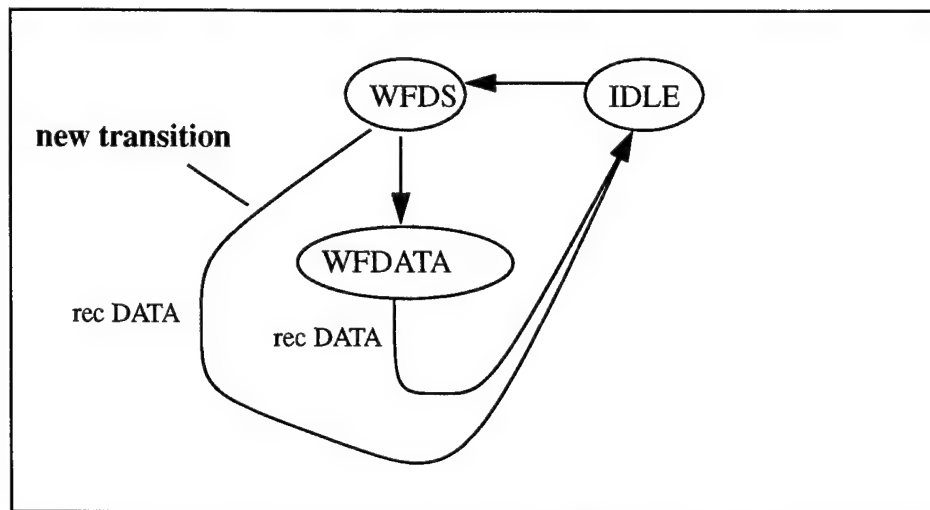
### **E. IMPROVEMENTS SUGGESTED BY ANALYSIS**

Three possible improvements to MACAW were suggested by the analysis. These were three additional transitions in the state diagram. The first was a transition from the WFDS state for receiving the DATA message without first receiving the DS message. The second transition suggested by the analysis was to add a loop transition to the WFDS that would allow receipt of a RTS message and respond with a CTS message. The third and last transition was similar in concept to the loop transiting in the WFDS state except this one would be added to the WFCTS state so that a RTS could be sent from that state. These possible improvements are discussed in greater detail below.

### 1. Receive DATA from WFDS State

The analysis showed that if the DS message was not received then even if the DATA message was received correctly it could not be processed. This would seem to be a waste of time given that both machines currently will return to the IDLE state and begin the message procedure over. Rather than reject the DATA message if the leading DS message is lost a transition is suggested from the WFDS state that allows receipt of the DATA message. This transition is depicted in the partial state diagram shown in Figure 7.

It might be argued that the likelihood is small of losing the DS message without also



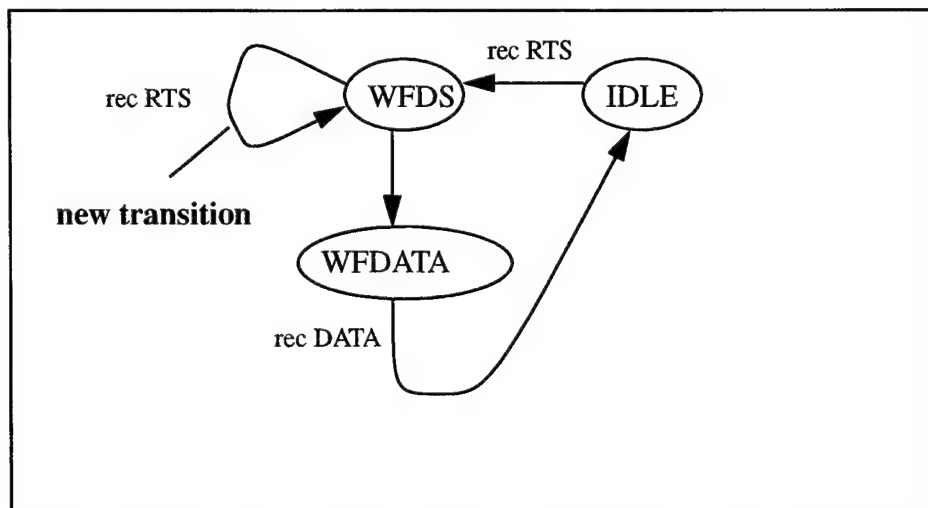
**Figure 7: Transition for receiving DATA from WFDS state**

losing the DATA message. Given the rather static and noise-free environment of the PARC LAN this is probably true. However, in an environment with highly mobile stations or a greater level of noise the possibility exists that a station might not receive the preceding RTS and CTS messages between machines 1 and 2 and consequently transmit a RTS at the same time as the DS message. Regardless of the situation that would cause loss of the DS message, the DATA message should be accepted if received properly.



## 2. Loop Transition in WFDS State

The second improvement suggested by the analysis was a loop transition to the WFDS state that would allow receipt of one or more RTS messages and respond with a CTS message. This situation would arise if the machine sending the RTS (machine 1) executes the timeout transition from the WFCTS state before receiving the CTS message from machine 2. The CTS may have been transmitted and lost or the original RTS may not have been received by machine 2. Rather than resume the media contention process, with the attendant delays, the machine might very well just accept a subsequent RTS. Rule 2 in the PAT of Table 3 would have to be modified to accommodate a timer value that is sufficient for whatever number of RTS messages the implementation wanted to receive before taking the timeout transition. This transition is depicted in Figure 8.

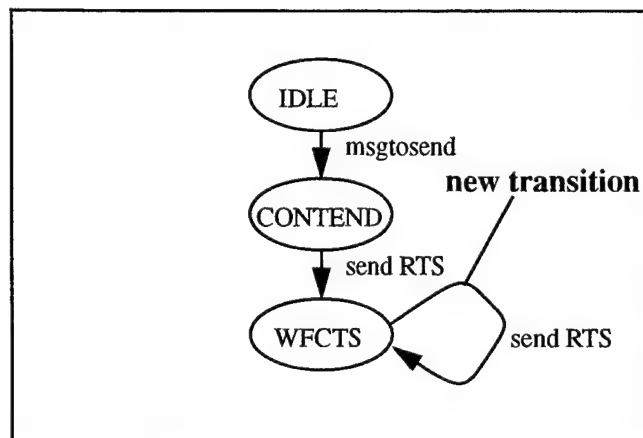


**Figure 8: Transition for RTS receipt in the WFDS state**

This transition would make little sense without the next transition that adds a send\_RTS loop to the WFCTS. Without the send\_RTS loop, the machine sending the RTS will timeout and resume the media contention process in spite of the receiving machine waiting in WFDS state.

### 3. Loop Transition in WFCTS State

Coupled with the previous transition is a transition that is a loop transition from the WFCTS state that sends one or more RTS messages. This allows the machine with the DATA message to continue to attempt to establish a connection with the intended receiver rather than timeout and resume the media contention process. The contention process might be rather lengthy depending on the value of the Backoff counter. This transition is depicted in the partial state diagram of Figure 9. The timer value of line 14 in the PAT of Table 3



**Figure 9: Transition for send RTS in the WFCTS state**

would have to be modified similarly to the rule 2 timer value to allow time for one or more RTS and CTS messages to be exchanged.



## VI. CONCLUSIONS

Wireless networking has enjoyed rapid growth in recent years as more and more applications are found that can benefit from wireless networks. Applications from inner-office LANs to WANs exist as well as private, commercial and military uses. One concern shared by all wireless networks is media access control (MAC). MAC protocols form a layer of any network model and are particularly important for wireless networking since the media, be it radio, light, or sound must be shared amongst the devices in the network. A good MAC protocol will resolve media contention both in the interests of fairness and network throughput.

Military Applications abound for single channel wireless radio protocols. Most currently use some form of Carrier Sense for media contention. As per [2] and [3] Carrier Sense is not an efficient means of media contention for wireless system due to the hidden and exposed nodes problem.

Packet radio networks first developed under the auspices of ARPA are now primarily a amateur radio province. However, with the advent of systems such as the Marine Corps Tactical Data Network (TDN), packet radio networks are seeing implementation as fielded operational systems with DOD units.

MACAW is a MAC protocol proposed by researchers at Xerox PARC that builds on the MACA protocol proposed by Karn. Both protocols approach the media contention problem in a similar manner. This approach, different than the more common CSMA approach, attempts to determine the state of the media at the intended receiver rather than just at the sender as in the CSMA approach. They do this by in essence reserving the media with both the intended receiver and other stations within hearing by exchanging RTS (Request to Send) and CTS (Clear to Send) messages.

This thesis presents a formal specification of MACAW using the formal model Systems of Communicating Machines. This formal specification was then encoded for use in an automated analysis of MACAW between 2, 3, and 4 machines.

## **A. FORMAL SPECIFICATION**

MACAW is formally specified and analyzed using a model called Systems of Communicating Machines. The Systems of Communicating Machines (SCM) model is used to assist in the describing and analyzing communications protocols. In specifying MACAW various additions to the original proposal [1] were added. These resulted from ambiguities or errors detected in the original proposal. Two new states and one new transition were added and two tinstones were renamed.

The specific changes added were: 1) adding the XMIT state to the transmitting side of the protocol that would mirror the WFDS and WFDATA states on the receive side, 2) adding the CONTEND\_2 state in place of the transition from the WFCONTEND state to the CONTEND state, 3) renaming two of the three timeout transitions to resolve ambiguity regarding which timeout was in question, and 4) adding a transition in the CONTEND state to receive a RRTS message instead of only allowing receipt of RRTS messages in IDLE state.

## **B. ANALYSIS**

An automated analysis of MACAW (with the changes mentioned above) was conducted. This automated analysis used an Ada program that uses the Systems of Communicating Machines model to analyze a given communication protocol. Each new protocol requires encoding definitions, predicates, actions, and output rules. The program is capable of conducting either global or system state analysis. The extent of the analysis may be varied by changing the input file that the executable reads. The input file may change numbers of machines, transitions, and initial states of the machines. A caveat is that any machines, transitions, or states in the input file must also be reflected in files for definitions, predicates, and actions. Essentially, once all possible machines, states, and transitions have been defined in the program then any of those items may be deleted from a subsequent analysis by removing them from the input file.

The actual analysis consisted of 2, 3, and 4 machines. Machines 1 and 2 were used to analyze the behavior of basic protocol without any transitions to the QUIET state. Machine 3 was added to cause machine 1 to exercise the transitions to the QUIET state. Machine 4 was added to analyze the hidden and exposed node performance. In every configuration the DATA messages were successfully exchanged. The number of states varied from 275 states for 2 machines to 1580 states for the four machine configuration.

### **C. IMPROVEMENTS TO MACAW**

The course of analysis suggested three improvements to MACAW. These improvements took the form of additional transitions to the state diagram. The first transition was a transition that would allow receipt of a DATA message in the WFDS state. In the original proposal the DS message must be received before the DATA message. The proposed addition would allow the DATA message to be received in the case where the DS message is not received (a collision with another message is one plausible cause). The second and third transitions are related in that in effect they give the two machines involved in a data exchange more than one chance to exchange RTS and CTS messages before taking the timeout transiting back to the IDLE state and resuming the media contention process. The first of these two transitions would allow the intended recipient of a DATA message to receive one or more RTS messages in the WFDS state. The second transition would allow the machine that has a DATA message for transfer to send one or more RTS messages to establish a connection before returning to the IDLE state via the timeout transition.

### **D. SUGGESTIONS FOR FURTHER WORK**

Several possible areas for further work exist. The ideal area for further work would be implementation of MACAW\_2 in a hardware system where it could undergo actual testing and use. Short of that goal, MACAW's applicability to packet radio networks can be explored. MACAW should be relevant to packet radio networks since MACAW's conceptual ancestor, MACA, was proposed for use in packet radio networks. Another area

of work could be redesign of the analysis program to incorporate runtime changes to the protocol under analysis. Currently, every change necessitates recompiling the code. Another useful feature for the analysis program would include more informative output. As is the output must be hand traced to gain information.

## LIST OF REFERENCES

1. Bharghavan, V., Demers, A., Shenker, S., Zhang, L., "MACAW: A Media Access Protocol for Wireless LAN's", Association for Computing Machinery SIGCOMM 94 - 8/94.
2. Karn, P., "MACA - A New Channel Access Method for Packet Radio", ARRL/CRRL Amateur Radio 9th Computer Networking Conference, September 22, 1990.
3. Rypinski, C., "Limitations of CSMA in 802.11 Radiolan Applications", IEEE 802.11 Working Group Paper 802.11/91-46a.
4. Katz, R., "Adaptation and Mobility in Wireless Information Systems", IEEE Personal Communications, First Quarter 1994.
5. Epstein, M., et al, "Application of Commercial Wireless LAN Technology to Forward Area Mobile Communication", Conference Proceedings Military Communications, MILCOMM 94-2.
6. Lundy, G., "Specification and analysis of a data transfer protocol using systems of communicating machines", Distributed Computing, May 1991.
7. Bulbul, B. "A Protocol Validator for the SCM and CFSM Models", Master's Thesis, Naval Postgraduate School, June 1993.





## APPENDIX

This appendix contains the source code used to translate the specification of MACAW into executable code for the automated analysis done in Chapter V.

```
with TEXT_IO;
use TEXT_IO;
package definitions is
  num_of_machines : constant := 4 ;
  type scm_transition_type is
    (msgtosend1,timeout1,send_RTS1,rcv_CTS1,rcv_ACK1,rcv_DS1,
     snd_data1,rcv_RTS1,send_CTS1,rcv_DATA1,rcv_RRTS1,
     quiet1, wfcontend1, contend1, send_rrts1, send_ACK1,

     msgtosend2,timeout2,send_RTS2,rcv_CTS2,rcv_ACK2,rcv_DS2,
     snd_data2,rcv_RTS2,send_CTS2,rcv_DATA2,rcv_RRTS2,
     quiet2, wfcontend2, contend2, send_rrts2, send_ACK2,

     msgtosend3,timeout3,send_RTS3,rcv_CTS3,rcv_ACK3,rcv_DS3,
     snd_data3,rcv_RTS3,send_CTS3,rcv_DATA3,rcv_RRTS3,
     quiet3, wfcontend3, contend3, send_rrts3, send_ACK3,

     msgtosend4,timeout4,send_RTS4,rcv_CTS4,rcv_ACK4,rcv_DS4,
     snd_data4,rcv_RTS4,send_CTS4,rcv_DATA4,rcv_RRTS4,
     quiet4, wfcontend4, contend4, send_rrts4, send_ACK4,

     unused);

  type buffer_type is (DS1, RTS1, CTS1, DATA1, ACK1, RRTS1,
    DS2, RTS2, CTS2, DATA2, ACK2, RRTS2,
    DS3, RTS3, CTS3, DATA3, ACK3, RRTS3,
    DS4, RTS4, CTS4, DATA4, ACK4, RRTS4,E );

  package buff_enum_io is new Enumeration_io(buffer_type);
  use buff_enum_io;
  type dummy_type is range 1..255;

  type machine1_state_type is
    record
```

```
    out_buff1 : buffer_type := DATA2; -- data message for m2
    in_buff1 : buffer_type:= E;
end record;
```

```
type machine2_state_type is
  record
    out_buff2,
    in_buff2 : buffer_type:= E;
  end record;
```

```
type machine3_state_type is
  record
    out_buff3 : buffer_type := DATA3;
    in_buff3 : buffer_type := E;
  end record;
```

```
type machine4_state_type is
  record
    out_buff4,
    in_buff4 : buffer_type:= E;
  end record;
```

```
type machine5_state_type is
  record
    dummy : dummy_type;
  end record;
```

```
type machine6_state_type is
  record
    dummy : dummy_type;
  end record;
```

```
type machine7_state_type is
  record
    dummy : dummy_type;
  end record;
```

```
type machine8_state_type is
  record
    dummy : dummy_type;
  end record;
```

```
type global_variable_type is
```

```
record
  CHAN1,
  CHAN2,
  CHAN3,
  CHAN4 : buffer_type := E;
  CHAN1_DA, CHAN2_DA, CHAN3_DA, CHAN4_DA : natural := 0;
end record;

end definitions;
```

```

procedure Analyze_Predicates_Machine1(local : machine1_state_type;
GLOBAL: global_variable_type;
s : natural;
w :in out transition_stack_package.stack) is
begin
case s is
when 0 =>
if (GLOBAL.CHAN1 = RTS1) AND (GLOBAL.CHAN1_DA = 1) then
if (LOCAL.in_buff1 = DATA1) then
Push (w, send_ACK1);
else
Push(w, rcv_RTS1);    --go to state 5
end if;
elsif (GLOBAL.CHAN1 = RRTS1) AND (GLOBAL.CHAN1_DA = 1)then
Push(w, rcv_RRTS1);    --go to state 2
elsif( LOCAL.out_buff1 /= E) then
Push(w,msgtosend1);    --go to state 1
elsif (GLOBAL.CHAN1 /= E) AND (GLOBAL.CHAN1_DA /= 1)then
Push(w, quiet1);    --go to state (Quiet)
end if;

when 1 =>    --corresponds to MACAW CONTENTEND state
if (GLOBAL.CHAN1 = RTS1) AND (GLOBAL.CHAN1_DA = 1) then
Push (w, rcv_RTS1);    -- go to state 5
elsif (GLOBAL.CHAN1 = RRTS1) AND (GLOBAL.CHAN1_DA = 1)then
Push(w, rcv_RRTS1);    --go to state 2
elsif (GLOBAL.CHAN1 /= E) AND (GLOBAL.CHAN1_DA /= 1)then
Push(w, quiet1);
else
Push(w,send_RTS1);    -- go to state 2
end if;

when 2 =>
if (GLOBAL.CHAN1 = CTS1) AND (GLOBAL.CHAN1_DA = 1) then
Push(w, rcv_CTS1);    -- go to state 3
elsif (GLOBAL.CHAN1 = ACK1) AND (GLOBAL.CHAN1_DA = 1) then
Push(w, rcv_ACK1);    -- go to state 0, message already received
elsif (GLOBAL.CHAN1 /= E) AND (GLOBAL.CHAN1_DA /= 1)then
Push(w, quiet1);
elsif (GLOBAL.CHAN1 /= CTS1) then
Push(w, timeout1);    -- go to state 0
end if;

```

```

when 3 =>          --an intermediate state after WFCTS
    Push(w, snd_data1);    --go to state 4

when 4 =>          --MACAW WFAACK state
    if (GLOBAL.CHAN1 = ACK1 ) AND (GLOBAL.CHAN1_DA = 1) then
        Push(w, rcv_ACK1);    -- go to state 0
    elsif (GLOBAL.CHAN1 /= E) AND (GLOBAL.CHAN1_DA /= 1) then
        Push(w, quiet1);
    elsif (GLOBAL.CHAN1 /= ACK1) then
        Push(w, timeout1);    --go to state 0
    end if;

-- states 5 and 6 model the WFDS and WFDATA states of MACAW
-- machines
when 5 =>  -- corresponds to WFDS state
    if (GLOBAL.CHAN1 = DS1) AND (GLOBAL.CHAN1_DA = 1) then
        Push(w, rcv_DS1);    -- go to state 6
    elsif (GLOBAL.CHAN1 /= E) AND (GLOBAL.CHAN1_DA /= 1) then
        Push(w, quiet1);
    elsif (GLOBAL.CHAN1 /= DS1) then
        Push(w, timeout1);    -- go to state 0
    end if;

when 6 =>  -- state 6 corresponds to WFDATA state
    if (GLOBAL.CHAN1 = DATA1) AND (GLOBAL.CHAN1_DA = 1) then
        Push(w, rcv_DATA1);    -- go to state 0
    elsif (GLOBAL.CHAN1 /= E) AND (GLOBAL.CHAN1_DA /= 1) then
        Push(w, quiet1);
    elsif (GLOBAL.CHAN1 /= DATA1) then
        Push(w, timeout1);    -- go to state 0
    end if;

when 7 =>  --corresponds to QUIET
    if (GLOBAL.CHAN1 = RTS1) AND (GLOBAL.CHAN1_DA = 1) then
        Push(w, wfcontend1);    --go to state 8
    else
        Push(w, timeout1);    -- go to state 0
    end if;

when 8 =>
    if (GLOBAL.CHAN1 /= E) AND (GLOBAL.CHAN1_DA /= 1) then
        Push(w, quiet1);    -- go to state 7
    end if;

```

```

else
    Push(w, contend1); -- go to state 9
end if;

when 9 =>
    if (GLOBAL.CHAN1 /= E) AND (GLOBAL.CHAN1_DA /= 1) then
        Push(w, quiet1); -- go to state 7
    else
        Push(w, send_rrts1); -- go to state 0
    end if;

when others =>
    null;
end case;
end Analyze_Predicates_Machine1;
-----
separate (main)
procedure Analyze_Predicates_Machine2(local : machine2_state_type;
    GLOBAL: global_variable_type;
    s: natural;
    w :in out transition_stack_package.stack) is
begin
    case s is
        when 0 =>
            if (GLOBAL.CHAN2 = RTS2) AND (GLOBAL.CHAN2_DA = 2) then
                if (LOCAL.in_buff2 = DATA2) then
                    Push (w, send_ACK2); -- Data was already received
                else
                    Push(w, rcv_RTS2); --go to state 5
                end if;
            elsif (GLOBAL.CHAN2 = RRTS2) AND (GLOBAL.CHAN2_DA = 2) then
                Push(w, rcv_RRTS2); -- go to state 2
            elsif (LOCAL.out_buff2 /= E) then
                Push(w,msgtosend2); -- go to state 1
            elsif (GLOBAL.CHAN2 /= E) AND (GLOBAL.CHAN2_DA /= 2) then
                Push(w, quiet2);
            end if;

            when 1 => --corresponds to MACAW CONTENTEND state
                if (GLOBAL.CHAN2 = RTS2) AND (GLOBAL.CHAN2_DA = 2) then
                    Push(w, rcv_RTS2); -- go to state 5
                elsif (GLOBAL.CHAN2 = RRTS2) AND (GLOBAL.CHAN2_DA = 2) then
                    Push(w, rcv_RRTS2); -- go to state 2

```

```

elseif (GLOBAL.CHAN2 /= E) AND (GLOBAL.CHAN2_DA /= 2) then
    Push(w, quiet2);
else
    Push(w, send_RTS2);    -- go to state 2
end if;

when 2 =>                -- 2 is WFCTS state
    if (GLOBAL.CHAN2 = CTS2) AND (GLOBAL.CHAN2_DA = 2) then
        Push(w, rcv_CTS2);    -- go to state 3
    elseif (GLOBAL.CHAN2 = ACK2) AND (GLOBAL.CHAN2_DA = 2) then
        Push(w, rcv_ACK2);    -- go to state 0, message already received
    elseif (GLOBAL.CHAN2 /= E) AND (GLOBAL.CHAN2_DA /= 2) then
        Push(w, quiet2);
    elseif (GLOBAL.CHAN2 /= CTS2) then
        Push(w, timeout2);    -- go to state 0
    end if;

when 3 =>                --an intermediate state after WFCTS
    Push(w, snd_data2);    -- go to state 4

when 4 =>                --MACAW WPACK state
    if (GLOBAL.CHAN2 = ACK2 ) AND (GLOBAL.CHAN2_DA = 2) then
        Push(w, rcv_ACK2);    -- go to state 0
    elseif (GLOBAL.CHAN2 /= E) AND (GLOBAL.CHAN2_DA /= 2) then
        Push(w, quiet2);
    elseif (GLOBAL.CHAN2 /= ACK2) then
        Push(w, timeout2);    -- go to state 0
    end if;

when 5 => --5 corresponds to WFDS
    if (GLOBAL.CHAN2 = DS2) AND (GLOBAL.CHAN2_DA = 2) then
        Push(w, rcv_DS2);    -- go to state 6
    elseif (GLOBAL.CHAN2 /= E) AND (GLOBAL.CHAN2_DA /= 2) then
        Push(w, quiet2);
    elseif (GLOBAL.CHAN2 /= DS2) then
        Push (w, timeout2);    -- go to state 0
    end if;

when 6 => --6 corresponds to WFDATA
    if (GLOBAL.CHAN2 = DATA2) AND (GLOBAL.CHAN2_DA = 2) then
        Push(w, rcv_DATA2);    -- go to state 0

```



```

    elsif (GLOBAL.CHAN2 /= E) AND (GLOBAL.CHAN2_DA /= 2) then
        Push(w, quiet2);
    elsif (GLOBAL.CHAN2 /= DATA2) then
        Push(w, timeout2); -- go to state 0
    end if;

when 7 => --corresponds to QUIET
    if (GLOBAL.CHAN2 = RTS2) AND (GLOBAL.CHAN2_DA = 2) then
        Push(w, wfcontend2); --go to state 8
    else
        Push(w, timeout2); -- go to state 0
    end if;

when 8 =>
    if (GLOBAL.CHAN2 /= E) AND (GLOBAL.CHAN2_DA /= 2) then
        Push(w, quiet2); -- go to state 7
    else
        Push(w, contend2); -- go to state 9
    end if;

when 9 =>
    if (GLOBAL.CHAN2 /= E) AND (GLOBAL.CHAN2_DA /= 2) then
        Push(w, quiet2); -- go to state 7
    else
        Push(w, send_rrts2); -- go to state 0
    end if;

when others =>
    null;
end case;
end Analyze_Predicates_Machine2;

```

---

```

separate (main)
procedure Analyze_Predicates_Machine3(local : machine3_state_type;
    GLOBAL: global_variable_type;
    s : natural;
    w : in out transition_stack_package.stack) is

begin
    case s is
        when 0 =>

```

```

if (GLOBAL.CHAN3 = RTS3) AND (GLOBAL.CHAN3_DA = 3) then
  if (LOCAL.in_buff3 = DATA3) then
    Push (w, send_ACK3);
  else
    Push(w, rcv_RTS3);      --go to state 5
  end if;
elsif (GLOBAL.CHAN3 = RRTS3) AND (GLOBAL.CHAN3_DA = 3) then
  Push(w, rcv_RRTS3); -- go to state 2
elsif( LOCAL.out_buff3 /= E) then
  Push(w,msgtosend3); -- go to state 1
elsif (GLOBAL.CHAN3 /= E)AND (GLOBAL.CHAN3_DA = 3) then
  Push(w, quiet3);      -- go to state 7
end if;

when 1 =>      --corresponds to MACAW CONTENTEND state
  if (GLOBAL.CHAN3 = RTS3) AND (GLOBAL.CHAN3_DA = 3) then
    Push(w, rcv_RTS3); -- go to state 5
  elsif (GLOBAL.CHAN3 /= E)AND (GLOBAL.CHAN3_DA = 3) then
    Push(w, quiet3);      -- go to state 7
  else
    Push(w,send_RTS3); -- go to state 2
  end if;

-- cases 2-4 added 10 Apr and 13 Apr added if clauses
when 2 =>      --WFCTS state
  if (GLOBAL.CHAN3 = CTS3) AND (GLOBAL.CHAN3_DA = 3) then
    Push(w, rcv_CTS3); -- go to state 3
  elsif (GLOBAL.CHAN3 = ACK3) AND (GLOBAL.CHAN3_DA = 3) then
    Push(w, rcv_ACK3 ); -- go to state 0, message already received
  elsif (GLOBAL.CHAN3 /= E)AND (GLOBAL.CHAN3_DA = 3) then
    Push(w, quiet3);      -- go to state 7
  elsif (GLOBAL.CHAN3 /= CTS3) then
    Push(w, timeout3); -- go to state 0
  end if;

when 3 =>      --an intermediate state after WFCTS
  Push(w, snd_data3); -- go to state 4

when 4 =>      --MACAW WFAACK state
  if (GLOBAL.CHAN3 = ACK3 ) AND (GLOBAL.CHAN3_DA = 3) then
    Push(w, rcv_ACK3); -- go to state 0
  elsif (GLOBAL.CHAN3 /= E)AND (GLOBAL.CHAN3_DA = 3) then
    Push(w, quiet3);      -- go to state 7

```

```

elseif (GLOBAL.CHAN3 /= ACK3) then
    Push(w, timeout3); -- go to state 0
end if;

when 5 => -- 5 is WFDS state
    if (GLOBAL.CHAN3 = DS3) AND (GLOBAL.CHAN3_DA = 3) then
        Push(w, rcv_DS3); -- go to state 6
    elseif (GLOBAL.CHAN3 /= E) AND (GLOBAL.CHAN3_DA = 3) then
        Push(w, quiet3); -- go to state 7
    elseif (GLOBAL.CHAN3 /= DS3) then
        Push(w, timeout3); -- go to state 0
    end if;

when 6 => -- 6 is WFDATA state
    if (GLOBAL.CHAN3 = DATA3) AND (GLOBAL.CHAN3_DA = 3) then
        Push(w, rcv_DATA2); -- go to state 0
    elseif (GLOBAL.CHAN3 /= E) AND (GLOBAL.CHAN3_DA /= 3) then
        Push(w, quiet3); -- go to state 7
    elseif (GLOBAL.CHAN3 /= DATA3) then
        Push(w, timeout3); -- go to state 0
    end if;

when 7 => -- corresponds to QUIET
    if (GLOBAL.CHAN3 = RTS3) AND (GLOBAL.CHAN3_DA = 3) then
        Push(w, wfcontend3); -- go to state 8
    else
        Push(w, timeout3); -- go to state 0
    end if;

when 8 =>
    if (GLOBAL.CHAN3 /= E) AND (GLOBAL.CHAN3_DA /= 3) then
        Push(w, quiet3); -- go to state 7
    else
        Push(w, contend3); -- go to state 9
    end if;

when 9 =>
    if (GLOBAL.CHAN3 /= E) AND (GLOBAL.CHAN3_DA /= 3) then
        Push(w, quiet3); -- go to state 7
    else
        Push(w, send_rrts3); -- go to state 0
    end if;

```

```

when others =>
    null;
end case;
end Analyze_Predicates_Machine3;

```

---

```

separate (main)
procedure Analyze_Predicates_Machine4(local : machine4_state_type;
    GLOBAL: global_variable_type;
    s : natural;
    w : in out transition_stack_package.stack) is

begin
    case s is
        when 0 =>
            if (GLOBAL.CHAN4 = RTS4) AND (GLOBAL.CHAN4_DA = 4) then
                if (LOCAL.in_buff4 = DATA4) then
                    Push (w, send_ACK4);
                else
                    Push(w, rcv_RTS4);    --go to state 5
                end if;
            elsif (GLOBAL.CHAN4 = RRTS4) AND (GLOBAL.CHAN4_DA = 4) then
                Push(w, rcv_RRTS4); -- go to state 2
            elsif( LOCAL.out_buff4 /= E) then
                Push(w,msgtosend4); -- go to state 1
            elsif (GLOBAL.CHAN4 /= E)AND (GLOBAL.CHAN4_DA = 4) then
                Push(w, quiet4);    -- go to state 7
            end if;

            when 1 =>                --corresponds to MACAW CONTEND state
                if (GLOBAL.CHAN4 = RTS4) AND (GLOBAL.CHAN4_DA = 4)  then
                    Push(w, rcv_RTS4);    -- go to state 5
                elsif (GLOBAL.CHAN4 /= E)AND (GLOBAL.CHAN4_DA = 4) then
                    Push(w, quiet4);    -- go to state 7
                else
                    Push(w,send_RTS4);    -- go to state 2
                end if;

            when 2 =>                --WFCTS state
                if (GLOBAL.CHAN4 = CTS4) AND (GLOBAL.CHAN4_DA = 4) then
                    Push(w, rcv_CTS3);    -- go to state 3
                end if;
    end case;
end Analyze_Predicates_Machine4;

```

```

elseif (GLOBAL.CHAN4 = ACK3) AND (GLOBAL.CHAN4_DA = 4) then
    Push(w, rcv_ACK3 );    -- go to state 0, message already received
elseif (GLOBAL.CHAN4 /= E)AND (GLOBAL.CHAN4_DA = 4) then
    Push(w, quiet4);    -- go to state 7
elseif (GLOBAL.CHAN4 /= CTS4) then
    Push(w, timeout4);    -- go to state 0
end if;

when 3 =>    --an intermediate state after WFCTS
    Push(w, snd_data4);    -- go to state 4

when 4 =>    --MACAW WFAACK state
    if (GLOBAL.CHAN4 = ACK4 ) AND (GLOBAL.CHAN4_DA = 4) then
        Push(w, rcv_ACK4);    -- go to state 0
    elseif (GLOBAL.CHAN4 /= E)AND (GLOBAL.CHAN4_DA = 4) then
        Push(w, quiet4);    -- go to state 7
    elseif (GLOBAL.CHAN4 /= ACK4) then
        Push(w, timeout4);    -- go to state 0
    end if;

when 5 =>    -- 5 is WFDS state
    if (GLOBAL.CHAN4 = DS4) AND (GLOBAL.CHAN4_DA = 4) then
        Push(w, rcv_DS4);    -- go to state 6
    elseif (GLOBAL.CHAN4 /= E)AND (GLOBAL.CHAN4_DA = 4) then
        Push(w, quiet4);    -- go to state 7
    elseif (GLOBAL.CHAN4 /= DS4) then
        Push (w, timeout4);    -- go to state 0
    end if;

when 6 =>    -- 6 is WFDATA state
    if (GLOBAL.CHAN4 = DATA4) AND (GLOBAL.CHAN4_DA = 4) then
        Push(w,rcv_DATA4);    -- go to state 0
    elseif (GLOBAL.CHAN4 /= E)AND (GLOBAL.CHAN4_DA = 4) then
        Push(w, quiet4);    -- go to state 7
    elseif (GLOBAL.CHAN4 /= DATA4) then
        Push(w, timeout4);    -- go to state 0
    end if;

when 7 =>    --corresponds to QUIET
    if (GLOBAL.CHAN4 = RTS4) AND (GLOBAL.CHAN4_DA = 4) then
        Push(w, wfcontend4);    --go to state 8
    else
        Push(w, timeout4);    -- go to state 0
    end if;

```

```

end if;

when 8 =>
  if (GLOBAL.CHAN4 /= E) AND (GLOBAL.CHAN4_DA /= 4) then
    Push(w, quiet4); -- go to state 7
  else
    Push(w, contend4); -- go to state 9
  end if;

when 9 =>
  if (GLOBAL.CHAN4 /= E) AND (GLOBAL.CHAN4_DA /= 4) then
    Push(w, quiet4); -- go to state 7
  else
    Push(w, send_rrts4); -- go to state 0
  end if;

when others =>
  null;
end case;
end Analyze_Predicates_Machine4;

```

---

```

-- the following Analyze Predicate procedures are placeholders
separate (main)
procedure Analyze_Predicates_Machine5(local : machine5_state_type;
GLOBAL: global_variable_type;
s : natural;
w : in out transition_stack_package.stack) is

```

```

begin
  null;
end Analyze_Predicates_Machine5;

```

---

```

separate (main)
procedure Analyze_Predicates_Machine6(local : machine6_state_type;
GLOBAL: global_variable_type;
s : natural;
w : in out transition_stack_package.stack) is

```

```

begin
  null;
end Analyze_Predicates_Machine6;

```

---

```
separate (main)
procedure Analyze_Predicates_Machine7(local : machine7_state_type;
    GLOBAL: global_variable_type;
    s : natural;
    w : in out transition_stack_package.stack) is

begin
    null;
end Analyze_Predicates_Machine7;
```

---

```
separate (main)
procedure Analyze_Predicates_Machine8(local : machine8_state_type;
    GLOBAL: global_variable_type;
    s : natural;
    w : in out transition_stack_package.stack) is

begin
    null;
end Analyze_Predicates_Machine8;
```

```

-----
separate (main)
procedure Action(in_system_state : in out Gstate_record_type;
                 in_transition   : in out scm_transition_type;
                 out_system_state : in out Gstate_record_type) is

begin
-- actions for machine 1

case (in_transition) is
when (msgtosend1) =>
    null;
    -- nothing is done when a message is added to the out buffer
    -- except move into the Contend state (state 1)

when (timeout1) =>
    __*****
    -- we dont want to clear the channel upon timeout
    -- out_system_state.GLOBAL_VARIABLES.CHAN2:= E;
    out_system_state.machine1_state.in_buff1 := E;

when (send_RTS1) =>
    out_system_state.GLOBAL_VARIABLES.CHAN2:= RTS2;
    out_system_state.GLOBAL_VARIABLES.CHAN2_DA := 2;
    --in_system_state.machine1_state.out_buff1;
    --out_system_state.machine1_state.out_buff1 := E;

when (rcv_CTS1) =>
    out_system_state.machine1_state.in_buff1 :=
        in_system_state.GLOBAL_VARIABLES.CHAN1;
    out_system_state.GLOBAL_VARIABLES.CHAN1 := E;
    out_system_state.GLOBAL_VARIABLES.CHAN2 := DS2;
    out_system_state.GLOBAL_VARIABLES.CHAN2_DA := 2;

when (snd_data1) =>
    out_system_state.GLOBAL_VARIABLES.CHAN2:=
        in_system_state.machine1_state.out_buff1;
    out_system_state.GLOBAL_VARIABLES.CHAN2_DA := 2;
    --out_system_state.machine1_state.out_buff1 := E;
    -- out_buff1 should keep DATA until 1 gets an ack
    -- out_buff1 := E commented out for 11-4 rgraph

```



```

when (rcv_ACK1) => -- go to state 0
    out_system_state.machine1_state.in_buff1 :=
        in_system_state.GLOBAL_VARIABLES.CHAN1;
    out_system_state.GLOBAL_VARIABLES.CHAN1:= E;
    out_system_state.machine1_state.out_buff1 := E;
-- sets out_buffer to empty and creates deadlock

when (rcv_RTS1) => -- go to state 5
    out_system_state.machine1_state.in_buff1 :=
        in_system_state.GLOBAL_VARIABLES.CHAN1;
    out_system_state.GLOBAL_VARIABLES.CHAN2 :=CTS2;
    out_system_state.GLOBAL_VARIABLES.CHAN2_DA := 2;
    out_system_state.GLOBAL_VARIABLES.CHAN1 := E;
-- outsystem state.global variables.chan2 := E uncommented
-- 11may because chan2 must be clear for mach 2 to transmit

when (rcv_DS1) =>
    out_system_state.machine1_state.in_buff1 :=
        in_system_state.GLOBAL_VARIABLES.CHAN1;
    out_system_state.GLOBAL_VARIABLES.CHAN1 :=E;

when (rcv_DATA1) =>
    out_system_state.machine1_state.in_buff1 :=
        in_system_state.GLOBAL_VARIABLES.CHAN1;
-- out_system_state.machine2_state.out_buff2 :=
-- out_system_state.machine2_state.in_buff2;
    out_system_state.GLOBAL_VARIABLES.CHAN1 :=E;
    out_system_state.GLOBAL_VARIABLES.CHAN2 :=ACK2;
    out_system_state.GLOBAL_VARIABLES.CHAN2_DA := 2;

when (send_ACK1) =>
    out_system_state.GLOBAL_VARIABLES.CHAN2:= ACK2;
    out_system_state.GLOBAL_VARIABLES.CHAN2_DA := 2;
-- out_system_state.machine1_state.out_buff1 := E;

when (quiet1) => -- go to state 7
    out_system_state.machine1_state.in_buff1 :=
        in_system_state.GLOBAL_VARIABLES.CHAN1;
-- this next action artificially clears machine =1's Channel
-- since normally the receiving machine would clear its own
-- channel. Artificial since 1 is not receiving the message
-- This is the only way 1 can ever transito to state 8 & 9.

```

```

    out_system_state.GLOBAL_VARIABLES.CHAN1 :=E;

when (wfcontend1) =>
    out_system_state.machine1_state.in_buff1 :=
        in_system_state.GLOBAL_VARIABLES.CHAN1;
    -- see comments for quiet1
    out_system_state.GLOBAL_VARIABLES.CHAN1 :=E;

when (contend1) =>
    out_system_state.machine1_state.in_buff1 :=
        in_system_state.GLOBAL_VARIABLES.CHAN1;
    -- see comments for quiet1
    out_system_state.GLOBAL_VARIABLES.CHAN1 :=E;

when (send_rrts1) =>
    out_system_state.GLOBAL_VARIABLES.CHAN2 := RRTS2;
    out_system_state.GLOBAL_VARIABLES.CHAN2_DA := 2;

when (rcv_RRTS1) =>
    out_system_state.GLOBAL_VARIABLES.CHAN2:= RTS2;
    out_system_state.GLOBAL_VARIABLES.CHAN2_DA := 2;

--
*****
--          transitions for machine 2
-----

when (msgtosend2) =>
    null;

when (timeout2) =>
    _*****
    -- out_system_state.GLOBAL_VARIABLES.CHAN1:= E;
    out_system_state.machine2_state.in_buff2 := E;

when (send_RTS2) =>
    out_system_state.GLOBAL_VARIABLES.CHAN1:= RTS1;
    out_system_state.GLOBAL_VARIABLES.CHAN1_DA := 1;
    --in_system_state.machine1_state.out_buff1;
    --out_system_state.machine1_state.out_buff1 := E;

when (rcv_CTS2) =>
    out_system_state.machine2_state.in_buff2 :=
        in_system_state.GLOBAL_VARIABLES.CHAN2;
    out_system_state.GLOBAL_VARIABLES.CHAN2 := E;

```

```

out_system_state.GLOBAL_VARIABLES.CHAN1 := DS1;
out_system_state.GLOBAL_VARIABLES.CHAN1_DA := 1;

when (snd_data2) =>
    out_system_state.GLOBAL_VARIABLES.CHAN1:=
        in_system_state.machine2_state.out_buff2;

when (rcv_ACK2) =>
    out_system_state.machine2_state.in_buff2 :=
        in_system_state.GLOBAL_VARIABLES.CHAN2;
    out_system_state.GLOBAL_VARIABLES.CHAN2:= E;
    out_system_state.machine2_state.out_buff2 := E;

when (rcv_RTS2) =>    -- go to state 5
    out_system_state.machine2_state.in_buff2 :=
        in_system_state.GLOBAL_VARIABLES.CHAN2;
    out_system_state.GLOBAL_VARIABLES.CHAN1 := CTS1;
    out_system_state.GLOBAL_VARIABLES.CHAN1_DA := 1;
    out_system_state.GLOBAL_VARIABLES.CHAN2 := E;
    -- outsystem state.global variables.chan2 := E uncommented
    -- 11may because chan2 must be clear for mach 2 to transmit

when (rcv_DS2) =>
    out_system_state.machine2_state.in_buff2 :=
        in_system_state.GLOBAL_VARIABLES.CHAN2;
    out_system_state.GLOBAL_VARIABLES.CHAN2 :=E;

when (rcv_DATA2) =>
    out_system_state.machine2_state.in_buff2 :=
        in_system_state.GLOBAL_VARIABLES.CHAN2;
    --once m2 gets data it sends data to m1
    out_system_state.machine2_state.out_buff2 := DATA1;
    out_system_state.GLOBAL_VARIABLES.CHAN2 :=E;
    out_system_state.GLOBAL_VARIABLES.CHAN1 := ACK1;
    out_system_state.GLOBAL_VARIABLES.CHAN1_DA := 1;

when (send_ACK2) =>
    out_system_state.GLOBAL_VARIABLES.CHAN2:= ACK1;
    out_system_state.GLOBAL_VARIABLES.CHAN1_DA := 1;

```

```

when (quiet2) => -- go to state 7
    out_system_state.machine2_state.in_buff2 :=
        in_system_state.GLOBAL_VARIABLES.CHAN2;
    -- this next action artificially clears machine 2 Channel
    -- since normally the receiving machine would clear its own
    -- channel. Artificial since 2 is not receiving the message
    -- This is the only way 2 can ever transiton to state 8 & 9.
    out_system_state.GLOBAL_VARIABLES.CHAN2 :=E;

when (wfcontend2) =>
    out_system_state.machine2_state.in_buff2 :=
        in_system_state.GLOBAL_VARIABLES.CHAN2;
    -- see comments for quiet2
    out_system_state.GLOBAL_VARIABLES.CHAN2 :=E;

when (contend2) =>
    out_system_state.machine2_state.in_buff2 :=
        in_system_state.GLOBAL_VARIABLES.CHAN2;
    -- see comments for quiet2
    out_system_state.GLOBAL_VARIABLES.CHAN2 :=E;

when (send_rrts2) =>
    out_system_state.GLOBAL_VARIABLES.CHAN1 := RRTS1;
    out_system_state.GLOBAL_VARIABLES.CHAN1_DA := 1;

when (rcv_RRTS2) =>
    out_system_state.GLOBAL_VARIABLES.CHAN1 := RTS1;
    out_system_state.GLOBAL_VARIABLES.CHAN1_DA := 1;

```

---

--transitions for machine3

```

when (msgtosend3) => --this needs to be modified from the same asbelow
    null;

```

```

when (timeout3) =>
    out_system_state.machine3_state.in_buff3 := E;

```

```

when (send_RTS3) =>
    out_system_state.GLOBAL_VARIABLES.CHAN1:= RTS4;
    out_system_state.GLOBAL_VARIABLES.CHAN1_DA := 4;
    out_system_state.GLOBAL_VARIABLES.CHAN4:= RTS4;
    out_system_state.GLOBAL_VARIABLES.CHAN4_DA := 4;

```

```

when (rcv_CTS3) =>
    out_system_state.machine3_state.in_buff3 :=
        in_system_state.GLOBAL_VARIABLES.CHAN3;
    out_system_state.GLOBAL_VARIABLES.CHAN3 := E;
    out_system_state.GLOBAL_VARIABLES.CHAN1 := DS4;
    out_system_state.GLOBAL_VARIABLES.CHAN1_DA := 4;
    out_system_state.GLOBAL_VARIABLES.CHAN4 := DS4;
    out_system_state.GLOBAL_VARIABLES.CHAN4_DA := 4;

```

```

when (snd_data3) =>
    out_system_state.GLOBAL_VARIABLES.CHAN1:=
        in_system_state.machine3_state.out_buff3;
    out_system_state.GLOBAL_VARIABLES.CHAN1_DA := 4;
    out_system_state.GLOBAL_VARIABLES.CHAN4:=
        in_system_state.machine3_state.out_buff3;
    out_system_state.GLOBAL_VARIABLES.CHAN4_DA := 4;

```

```

when (rcv_ACK3) =>
    out_system_state.machine3_state.in_buff3 :=
        in_system_state.GLOBAL_VARIABLES.CHAN3;
    out_system_state.GLOBAL_VARIABLES.CHAN3:= E;
    out_system_state.machine3_state.out_buff3 := E;

```

```

when (rcv_RTS3) =>    -- go to state 5
    out_system_state.machine3_state.in_buff3 :=
        in_system_state.GLOBAL_VARIABLES.CHAN3;
    out_system_state.GLOBAL_VARIABLES.CHAN1 := CTS4;
    out_system_state.GLOBAL_VARIABLES.CHAN1_DA := 4;
    out_system_state.GLOBAL_VARIABLES.CHAN4 := CTS4;
    out_system_state.GLOBAL_VARIABLES.CHAN4_DA := 4;
    out_system_state.GLOBAL_VARIABLES.CHAN3 := E;

```

```

when (rcv_DS3) =>
    out_system_state.machine3_state.in_buff3 :=
        in_system_state.GLOBAL_VARIABLES.CHAN3;
    out_system_state.GLOBAL_VARIABLES.CHAN3 :=E;

```

```

when (rcv_DATA3) =>
    out_system_state.machine3_state.in_buff3 :=

```

```

    in_system_state.GLOBAL_VARIABLES.CHAN3;
    out_system_state.GLOBAL_VARIABLES.CHAN3 := E;
    out_system_state.GLOBAL_VARIABLES.CHAN1 := ACK4;
    out_system_state.GLOBAL_VARIABLES.CHAN1_DA := 4;
    out_system_state.GLOBAL_VARIABLES.CHAN4 := ACK4;
    out_system_state.GLOBAL_VARIABLES.CHAN4_DA := 4;

when (send_ACK3) =>
    out_system_state.GLOBAL_VARIABLES.CHAN1:= ACK4;
    out_system_state.GLOBAL_VARIABLES.CHAN1_DA := 4;
    out_system_state.GLOBAL_VARIABLES.CHAN4:= ACK4;
    out_system_state.GLOBAL_VARIABLES.CHAN4_DA := 4;
    out_system_state.machine3_state.out_buff3 := E;

when (quiet3) => -- go to state 7
    out_system_state.machine3_state.in_buff3 :=
        in_system_state.GLOBAL_VARIABLES.CHAN3;

when (wfcontend3) =>
    out_system_state.machine3_state.in_buff3 :=
        in_system_state.GLOBAL_VARIABLES.CHAN3;

when (contend3) =>
    out_system_state.machine3_state.in_buff3 :=
        in_system_state.GLOBAL_VARIABLES.CHAN3;

when (send_rrts3) =>
    out_system_state.GLOBAL_VARIABLES.CHAN1 := RRTS4;
    out_system_state.GLOBAL_VARIABLES.CHAN1_DA := 4;
    out_system_state.GLOBAL_VARIABLES.CHAN4 := RRTS4;
    out_system_state.GLOBAL_VARIABLES.CHAN4_DA := 4;

```

---

```

--transitions for machine 4

```

```

    when (msgtosend4) =>
        null;

    when (timeout4) =>
        -- out_system_state.GLOBAL_VARIABLES.CHAN3:= E;
        out_system_state.machine4_state.in_buff4 := E;

    when (send_RTS4) =>
        out_system_state.GLOBAL_VARIABLES.CHAN3:= RTS3;

```

```

out_system_state.GLOBAL_VARIABLES.CHAN3_DA := 3;
out_system_state.GLOBAL_VARIABLES.CHAN2:= RTS3;
out_system_state.GLOBAL_VARIABLES.CHAN2_DA := 3;

when (rcv_CTS4) =>
    out_system_state.machine4_state.in_buff4 :=
        in_system_state.GLOBAL_VARIABLES.CHAN4;
    out_system_state.GLOBAL_VARIABLES.CHAN4 := E;
    out_system_state.GLOBAL_VARIABLES.CHAN3 := DS3;
    out_system_state.GLOBAL_VARIABLES.CHAN3_DA := 3;

when (snd_data4) =>
    out_system_state.GLOBAL_VARIABLES.CHAN3:=
        in_system_state.machine4_state.out_buff4;
    out_system_state.GLOBAL_VARIABLES.CHAN3_DA := 4;

when (rcv_ACK4) =>
    out_system_state.machine4_state.in_buff4 :=
        in_system_state.GLOBAL_VARIABLES.CHAN4;
    out_system_state.GLOBAL_VARIABLES.CHAN4:= E;
    out_system_state.machine4_state.out_buff4 := E;

when (rcv_RTS4) =>    -- go to state 5
    out_system_state.machine4_state.in_buff4 :=
        in_system_state.GLOBAL_VARIABLES.CHAN4;
    out_system_state.GLOBAL_VARIABLES.CHAN3 := CTS3;
    out_system_state.GLOBAL_VARIABLES.CHAN3_DA := 3;
    out_system_state.GLOBAL_VARIABLES.CHAN4 := E;

when (rcv_DS4) =>
    out_system_state.machine4_state.in_buff4 :=
        in_system_state.GLOBAL_VARIABLES.CHAN4;
    out_system_state.GLOBAL_VARIABLES.CHAN4 :=E;

when (rcv_DATA4) =>
    out_system_state.machine4_state.in_buff4 :=
        in_system_state.GLOBAL_VARIABLES.CHAN4;
    out_system_state.GLOBAL_VARIABLES.CHAN4 :=E;
    out_system_state.GLOBAL_VARIABLES.CHAN3 := ACK3;
    out_system_state.GLOBAL_VARIABLES.CHAN3_DA := 3;

```

```

when (send_ACK4) =>
  out_system_state.GLOBAL_VARIABLES.CHAN3:= ACK3;
  out_system_state.GLOBAL_VARIABLES.CHAN3_DA := 3;

when (quiet4) => -- go to state 7
  out_system_state.machine4_state.in_buff4 :=
    in_system_state.GLOBAL_VARIABLES.CHAN4;
  out_system_state.GLOBAL_VARIABLES.CHAN4 :=E;

when (wfcontend4) =>
  out_system_state.machine4_state.in_buff4 :=
    in_system_state.GLOBAL_VARIABLES.CHAN4;
  out_system_state.GLOBAL_VARIABLES.CHAN4 :=E;

when (contend4) =>
  out_system_state.machine4_state.in_buff4 :=
    in_system_state.GLOBAL_VARIABLES.CHAN4;
  out_system_state.GLOBAL_VARIABLES.CHAN4 :=E;

when (send_rrts4) =>
  out_system_state.GLOBAL_VARIABLES.CHAN3 := RRTS4;
  out_system_state.GLOBAL_VARIABLES.CHAN3_DA := 3;

when others =>
  put_line("There is an error in the Action procedure");
end case;
end Action;

```



```

separate (main)
procedure output_Gtuple(tuple : in out Gstate_record_type) is
begin
  if print_header then
    new_line(2);
    set_col(5);
    put_line(" m1(in_buff1,out_buff1),m2(in_buff2,out_buff2),m3(in_buff3,out_buff3),
              (CHAN1,CHAN2,CHAN3)");
    print_header := false;
  else
    put(" [“ & integer'image(tuple.machine_state(1)) );
    put(" , “);
    buff_enum_io.put(tuple.machine1_state.in_buff1);
    put(" , “);
    buff_enum_io.put(tuple.machine1_state.out_buff1);
    put(" ,” & integer'image(tuple.machine_state(2)) );
    put(" , “);
    buff_enum_io.put(tuple.machine2_state.in_buff2);
    put(" , “);
    buff_enum_io.put(tuple.machine2_state.out_buff2);
    put(" , “);
    put(integer'image(tuple.machine_state(3)) );
    put(" , “);
    buff_enum_io.put(tuple.machine3_state.in_buff3);
    put(" , “);
    buff_enum_io.put(tuple.machine3_state.out_buff3);
    put(" , “);
    buff_enum_io.put(tuple.GLOBAL_VARIABLES.CHAN1);
    put(" , “);
    buff_enum_io.put(tuple.GLOBAL_VARIABLES.CHAN2);
    put(" , “);
    buff_enum_io.put(tuple.GLOBAL_VARIABLES.CHAN3);
    put(" ]”);
  end if;

end output_Gtuple;

```

The following three pages are the input files used to analyze two, three and four machine configurations.

**One Machine input file:**

```
state 7
trans wfcontend1 8
trans timeout1 0
state 8
trans quiet1 7
trans contend1 9
state 9
trans send_rrts1 0
machine 2
state 0
trans msgtosend2 1
trans rcv_RTS2 5
trans rcv_RRTS2 2
trans send_ACK2 0
trans quiet2 7
state 1
trans send_RTS2 2
trans rcv_RTS2 5
trans quiet2 7
state 2
trans rcv_CTS2 3
trans timeout2 0
trans rcv_ACK2 0
trans quiet2 7
state 3
trans snd_data2 4
state 4
trans rcv_ACK2 0
trans timeout2 0
trans quiet2 7
state 5
trans rcv_DS2 6
trans timeout2 0
trans quiet2 7
state 6
trans rcv_DATA2 0
trans timeout2 0
```

trans quiet2 7  
state 7  
trans wfcontend2 8  
trans timeout2 0  
state 8  
trans quiet2 7  
trans contend2 9  
state 9  
trans send\_rrts2 0  
initial\_state 0 0  
finish

### Three machine input file:

```
start
number_of_machines 3
machine 1
state 0
trans msgtosend1 1
trans rcv_RTS1 5
trans rcv_RRTS1 2
trans send_ACK1 0
trans quiet1 7
state 1
trans send_RTS1 2
trans rcv_RTS1 5
trans quiet1 7
state 2
trans rcv_CTS1 3
trans quiet1 7
trans rcv_ACK1 0
trans timeout1 0
state 3
trans snd_data1 4
state 4
trans rcv_ACK1 0
trans timeout1 0
trans quiet1 7
state 5
trans rcv_DS1 6
trans timeout1 0
trans quiet1 7
state 6
trans rcv_DATA1 0
trans timeout1 0
trans quiet1 7
state 7
trans wfcontend1 8
trans timeout1 0
state 8
trans quiet1 7
trans contend1 9
state 9
trans send_rrts1 0
machine 2
```

state 0  
trans msgtosend2 1  
trans rcv\_RTS2 5  
trans rcv\_RRTS2 2  
trans send\_ACK2 0  
trans quiet2 7  
state 1  
trans send\_RTS2 2  
trans rcv\_RTS2 5  
trans quiet2 7  
state 2  
trans rcv\_CTS2 3  
trans timeout2 0  
trans rcv\_ACK2 0  
trans quiet2 7  
state 3  
trans snd\_data2 4  
state 4  
trans rcv\_ACK2 0  
trans timeout2 0  
trans quiet2 7  
state 5  
trans rcv\_DS2 6  
trans timeout2 0  
trans quiet2 7  
state 6  
trans rcv\_DATA2 0  
trans timeout2 0  
trans quiet2 7  
state 7  
trans wfcontend2 8  
trans timeout2 0  
state 8  
trans quiet2 7  
trans contend2 9  
state 9  
trans send\_rrts2 0  
machine 3  
state 0  
trans msgtosend3 1  
trans rcv\_RTS3 5  
trans rcv\_RRTS3 2  
trans quiet3 7

state 1  
trans send\_RTS3 2  
trans rcv\_RTS3 5  
state 2  
trans rcv\_CTS3 3  
trans timeout3 0  
state 3  
trans snd\_data3 4  
state 4  
trans rcv\_ACK3 0  
trans timeout3 0  
state 5  
trans rcv\_DS3 6  
trans timeout3 0  
state 6  
trans rcv\_DATA3 0  
trans timeout3 0  
state 7  
trans wfcontend3 8  
trans timeout3 0  
state 8  
trans quiet3 7  
trans contend3 9  
state 9  
trans send\_rrts3 0  
initial\_state 0 7 0  
finish

#### Four Machine input file:

```
start
number_of_machines 4
machine 1
state 0
trans msgtosend1 1
trans rcv_RTS1 5
trans rcv_RRTS1 2
trans send_ACK1 0
trans quiet1 7
state 1
trans send_RTS1 2
trans rcv_RRTS1 2
trans rcv_RTS1 5
trans quiet1 7
state 2
trans rcv_CTS1 3
trans quiet1 7
trans rcv_ACK1 0
trans timeout1 0
state 3
trans snd_data1 4
state 4
trans rcv_ACK1 0
trans timeout1 0
trans quiet1 7
state 5
trans rcv_DS1 6
trans timeout1 0
trans quiet1 7
state 6
trans rcv_DATA1 0
trans timeout1 0
trans quiet1 7
state 7
trans wfcontend1 8
trans timeout1 0
state 8
trans quiet1 7
trans contend1 9
state 9
```

trans send\_rrts1 0  
machine 2  
state 0  
trans msgtosend2 1  
trans rcv\_RTS2 5  
trans rcv\_RRTS2 2  
trans send\_ACK2 0  
trans quiet2 7  
state 1  
trans send\_RTS2 2  
trans rcv\_RRTS2 2  
trans rcv\_RTS2 5  
trans quiet2 7  
state 2  
trans rcv\_CTS2 3  
trans timeout2 0  
trans rcv\_ACK2 0  
trans quiet2 7  
state 3  
trans snd\_data2 4  
state 4  
trans rcv\_ACK2 0  
trans timeout2 0  
trans quiet2 7  
state 5  
trans rcv\_DS2 6  
trans timeout2 0  
trans quiet2 7  
state 6  
trans rcv\_DATA2 0  
trans timeout2 0  
trans quiet2 7  
state 7  
trans wfcontend2 8  
trans timeout2 0  
state 8  
trans quiet2 7  
trans contend2 9  
state 9  
trans send\_rrts2 0  
machine 3  
state 0  
trans msgtosend3 1



trans rcv\_RTS3 5  
trans rcv\_RRTS3 2  
trans quiet3 7  
state 1  
trans send\_RTS3 2  
trans rcv\_RTS3 5  
state 2  
trans rcv\_CTS3 3  
trans timeout3 0  
state 3  
trans snd\_data3 4  
state 4  
trans rcv\_ACK3 0  
trans timeout3 0  
state 5  
trans rcv\_DS3 6  
trans timeout3 0  
state 6  
trans rcv\_DATA3 0  
trans timeout3 0  
state 7  
trans wfcontend3 8  
trans timeout3 0  
state 8  
trans quiet3 7  
trans contend3 9  
state 9  
trans send\_rrts3 0  
machine 4  
state 0  
trans msgtosend4 1  
trans rcv\_RTS4 5  
trans rcv\_RRTS4 2  
trans quiet4 7  
state 1  
trans send\_RTS4 2  
trans rcv\_RTS4 5  
state 2  
trans rcv\_CTS4 3  
trans timeout4 0  
state 3  
trans snd\_data4 4  
state 4

trans rcv\_ACK4 0  
trans timeout4 0  
state 5  
trans rcv\_DS4 6  
trans timeout4 0  
state 6  
trans rcv\_DATA4 0  
trans timeout4 0  
state 7  
trans wfcontend4 8  
trans timeout4 0  
state 8  
trans quiet4 7  
trans contend4 9  
state 9  
trans send\_rrts4 0  
initial\_state 0 0 0 0  
finish



## INITIAL DISTRIBUTION LIST

- |   |   |
|---|---|
| 1. Defense Technical Information Center<br>Cameron Station<br>Alexandria, VA 22304-6145                           | 2 |
| 2. Dudley Knox Library<br>Code 013<br>Naval Postgraduate School<br>Monterey, CA 93943-5002                        | 2 |
| 3. Director, Training and Education<br>MCCDC, Code C46<br>1019 Elliot Rd.<br>Quantico, Virginia 22134-5027        | 1 |
| 4. Chairman, Code CS<br>Computer Science Department<br>Naval Postgraduate School<br>Monterey, CA 93943            | 2 |
| 5. Dr Gilbert Lundy, Code CS/LN<br>Computer Science Department<br>Naval Postgraduate School<br>Monterey, CA 93943 | 2 |
| 6. Lou Stevens, Code CS/ST<br>Computer Science Department<br>Naval Postgraduate School<br>Monterey, CA 93943      | 1 |